BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

PATENT ABSTRACTS OF JAPAN

AA

(11)Publication number : 10-247172
(43)Date of publication of application : 14.09.1998

(51)Int.Cl. G06F 13/36

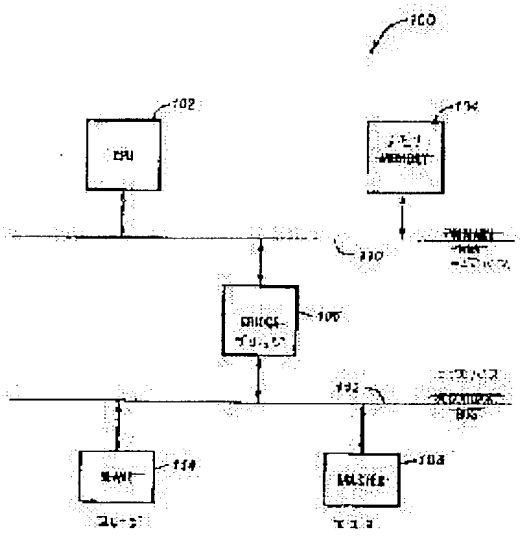
(21)Application number : 10-031923
(22)Date of filing : 05.01.1998

(71)Applicant : COMPAQ COMPUTER CORP
(72)Inventor : GOODRUM ALAN L

(30)Priority
Priority number : 96 774746 Priority date : 31.12.1996 Priority country : US

(54) BUFFER RESERVATION METHOD FOR BUS BRIDGE SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To attain the optimum transfer of data between buses by receiving a reservation command from a bus master and reserving a data buffer for the bus master in response to the reservation command.
SOLUTION: A bridge 106 includes a mechanism to reserve a data buffer. Thus, a bus master 108 connected to the bridge 106 or an agent requests the bridge 106 to reserve a buffer for the data read transfer that is intended by the master 108. Then the bridge 106 gives a common interface to the device such as the master 108 existing on a secondary bus 112 and attains the communication with the device such as a memory 104 existing on a primary bus 110. The master 108 also can communicate with other secondary bus devices such as a bus slave 114, etc., without requesting any band at all against the bus 110.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平10－247172

(43)公開日　平成10年(1998)９月14日

(51)Int.Cl.[6]	識別記号	Ｆ Ｉ	
Ｇ０６Ｆ　13/36	３１０	Ｇ０６Ｆ　13/36	３１０Ｆ

審査請求　未請求　請求項の数36　ＯＬ　外国語出願　（全 76 頁）

(21)出願番号　特願平10－31923

(22)出願日　平成10年(1998)１月５日

(31)優先権主張番号　７７４７４６

(32)優先日　1996年12月31日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　591030868
コンパック・コンピューター・コーポレーション
COMPAQ COMPUTER CORPORATION
アメリカ合衆国テキサス州77070，ヒューストン，ステイト・ハイウェイ　249，20555

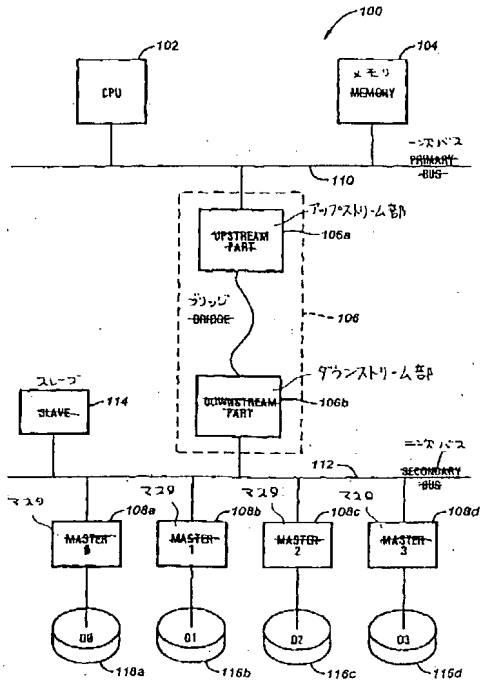
(72)発明者　アラン・エル・グッドラム
アメリカ合衆国テキサス州77375，トムボール，エイヴンフィールド　16522

(74)代理人　弁理士　社本　一夫　（外６名）

最終頁に続く

(54)【発明の名称】　バス・ブリッジ・システムのためのバッファ・リザベーション方法

(57)【要約】
【課題】　バス・ブリッジを通じたデータ転送方法を提供する。
【解決手段】　バス・ブリッジは、データを格納し、データを先取りし、データのライト・ポスティングを行うための多数のデータ・バッファを含む。バス・ブリッジと通信するデバイスは、２つの予約機構の１つによってバッファを予約することができる。予約機構は、バス・ブリッジに、アドレスおよびバイト・カウントを供給する。また、予約はいずれの上流側バス・ブリッジにも送出することができる。予約されたバッファは、バス・アクセスの効率的な使用のために先取りされる。バッファが予約されていない場合、データの先取りおよびフラッシュは、別のアルゴリズムにしたがって行われる。



【特許請求の範囲】
【請求項１】　バス・ブリッジとバス・マスタの間における通信方法であって、前記バス・ブリッジは少なくとも１つのデータ・バッファを有する、方法において、
（ａ）前記バス・マスタから予約コマンドを受け取るステップと、
（ｂ）前記予約コマンドに応答して、前記バス・マスタのためにデータ・バッファを予約するステップと、
を備える方法。
【請求項２】　請求項１記載の方法において、（ｃ）ステップ（ａ）の後に、バス・コマンドを受け取るステップを更に備える方法。
【請求項３】　請求項２記載の方法において、前記バス・コマンドは開始アドレスを含み、前記バス・コマンドおよび前記予約コマンドは共になってアドレス範囲を定義する、方法。
【請求項４】　請求項３記載の方法において、前記予約コマンドはカウントを含む、方法。
【請求項５】　請求項３記載の方法において、前記バス・コマンドがリード動作である場合に、
（ｄ）前記リード動作に従ってデータを取り込むステップであって、該データは予約された前記バッファを通じて受け渡される、ステップと、
（ｅ）前記リード動作および前記アドレス範囲に従ってデータを先取りするステップであって、先取りした該データは前記の予約されたデータ・バッファに書き込まれる、ステップと、
（ｆ）前記の取り込んだデータおよび先取りしたデータを前記バス・マスタに供給するステップと、
を更に備える方法。
【請求項６】　請求項５記載の方法において、
（ｇ）前記アドレス範囲を使い切るまで、ステップ（ｅ）および（ｆ）を繰り返すステップと、
（ｈ）ステップ（ｇ）の後にデータ・バッファ予約を取り消すステップと、
を更に備える方法。
【請求項７】　請求項１記載の方法において、前記予約コマンドは、フレーム信号がアサートされた後に受け取られる、方法。
【請求項８】　請求項１記載の方法において、前記バス・ブリッジおよび前記バス・マスタは、周辺素子相互接続（ＰＣＩ）バスに結合するためのものである、方法。
【請求項９】　第１と第２バス・ブリッジの間における通信の方法であって、前記第２バス・ブリッジは前記第１バス・ブリッジとバス・マスタとの間を結合するものである、方法において、
（ａ）前記第２バス・ブリッジにおいて前記バス・マスタから予約コマンドを受け取るステップと、
（ｂ）前記予約コマンドを前記第１バス・ブリッジに送出するステップと、を備える方法。
【請求項１０】　請求項９記載の方法において、前記第２バス・ブリッジは、予約コマンドが送出されるか否かを選択するためのプログラム可能な送出ビットを含み、更に、
（ｃ）ステップ（ｂ）の前に予約コマンドの送出をイネーブルするために前記送出ビットがセットされているか否かについて判定を行うステップを含み、
ステップ（ｂ）は、送出をイネーブルとするために前記送出ビットがセットされることを条件とする、
方法。
【請求項１１】　請求項９記載の方法において、
（ｄ）ステップ（ａ）の後にバス・コマンドを受け取るステップと、
（ｅ）前記予約コマンドおよび前記バス・コマンドに応答して、前記バス・マスタのためにデータ・バッファを予約するステップと、
を更に備える方法。
【請求項１２】　請求項１１記載の方法において、前記バス・コマンドは開始アドレスを含み、前記バス・コマンドおよび前記予約コマンドは共になってアドレス範囲を定義する、方法。
【請求項１３】　請求項１２記載の方法において、前記予約コマンドはカウントを含む、方法。
【請求項１４】　請求項１２記載の方法において、前記バス・コマンドがリード動作である場合に、
（ｆ）前記リード動作に従ってデータを取り込むステップであって、該データは予約された前記バッファを通じて渡される、ステップと、
（ｇ）前記リード動作および前記アドレス範囲に従ってデータを先取りするステップであって、先取りした該データは前記の予約したデータ・バッファに書き込まれる、ステップと、
（ｈ）前記の取り込んだデータおよび先取りしたデータを前記バス・マスタに供給するステップと、
を更に備える方法。
【請求項１５】　請求項１４記載の方法において、
（ｉ）前記アドレス範囲を使い切るまでステップ（ｇ）および（ｈ）を繰り返すステップと、
（ｊ）ステップ（ｉ）の後に前記のデータ・バッファ予約を取り消すステップと、
を更に備える方法。
【請求項１６】　請求項９記載の方法において、前記予約コマンドは、フレーム信号をアサートした後に受け取られる、方法。
【請求項１７】　請求項９記載の方法において、前記バス・ブリッジおよび前記バス・マスタは、周辺素子相互接続（ＰＣＩ）バスに結合するためのものである、方法。
【請求項１８】　バス・ブリッジにおいてバッファを予約する方法であって、

（ａ）フレーム指示を与えるステップと、
（ｂ）前記フレーム指示をあたえるときに予約コマンドを与えるステップと、
（ｃ）ステップ（ｂ）の後でかつ前記フレーム指示を与えている間に、バス・コマンドを与えるステップと、
を備える方法。
【請求項１９】　請求項１８記載の方法において、前記バス・コマンドは開始アドレスを含み、前記バス・コマンドおよび前記予約コマンドは共になってアドレス範囲を定義する、方法。
【請求項２０】　請求項１８記載の方法において、前記予約コマンドはカウントを含む、方法。
【請求項２１】　コンピュータ・システムであって、
主メモリと、
前記主メモリに結合されるプロセッサと、
大容量記憶システムと、
前記大容量記憶システムに結合されるバス・マスタであって、予約コマンドおよびバス・コマンドを含むバス動作を与えるように動作可能なバス・マスタと、
前記主メモリと前記バス・マスタの間に結合されるバス・ブリッジであって、少なくとも１つの割り当て可能なデータ・バッファを有し、前記バス動作を受け取り、該バス動作に応答して、前記バス・マスタの排他的使用のためにデータ・バッファを予約するように動作可能であるバス・ブリッジと、
を備えるコンピュータ・システム。
【請求項２２】　請求項２１記載のコンピュータ・システムにおいて、前記バス・コマンドは開始アドレスを含み、前記バス・コマンドおよび前記予約コマンドは共になってアドレス範囲を定義する、コンピュータ・システム。
【請求項２３】　請求項２２記載のコンピュータ・システムにおいて、前記予約コマンドはカウントを含む、コンピュータ・システム。
【請求項２４】　請求項２２記載のコンピュータ・システムにおいて、前記バス・コマンドがリード動作である場合に、
前記バス・ブリッジは、前記リード動作に従ってデータを取り込み、前記アドレス範囲に従ってデータを先取りし、データは要求されたときに前記バス・マスタに供給される、コンピュータ・システム。
【請求項２５】　請求項２４記載のコンピュータ・システムにおいて、前記バス・マスタが前記アドレス範囲の最後のアドレスにおいて読み取りを行うときに、前記バス・ブリッジは前記データ・バッファの予約を取り消す、コンピュータ・システム。
【請求項２６】　請求項２１記載のコンピュータ・システムにおいて、前記予約コマンドは、フレーム信号がアサートされた後に受け取られる、コンピュータ・システム。
【請求項２７】　請求項２１記載のコンピュータ・システムにおいて、前記バス・ブリッジおよび前記バス・マスタは周辺素子相互接続（ＰＣＩ）バスに結合される、コンピュータ・システム。
【請求項２８】　請求項２１記載のコンピュータ・システムにおいて、前記バス動作は、前記大容量記憶システムに書き込むためのメモリ・リード動作を含む、コンピュータ・システム。
【請求項２９】　コンピュータ・システムであって、
主メモリと、
前記主メモリに結合されるプロセッサと、
大容量記憶システムと、
前記大容量記憶システムに結合されるバス・マスタであって、予約コマンドとバス・コマンドとを含むバス動作を与えるように動作可能なバス・マスタと、
前記主メモリに結合される第１バス・ブリッジと、
前記第１バス・ブリッジと前記バス・マスタとの間に結合される第２バス・ブリッジであって、少なくとも１つの割り当て可能なデータ・バッファと送出ビットとを有し、前記バス動作を受け取り且つ前記バス動作に応答して前記バス・マスタの排他的使用のためにデータ・バッファを予約するように動作可能であり、前記送出ビットがセットされている場合に前記予約コマンドを前記第１ブリッジに送出するように動作可能である、第２バス・ブリッジと、
を備えるコンピュータ・システム。
【請求項３０】　請求項２９記載のコンピュータ・システムにおいて、前記バス・コマンドは開始アドレスを含み、前記バス・コマンドおよび前記予約コマンドは共になってアドレス範囲を定義する、コンピュータ・システム。
【請求項３１】　請求項３０記載のコンピュータ・システムにおいて、前記予約コマンドはカウントを含む、コンピュータ・システム。
【請求項３２】　請求項２９記載のコンピュータ・システムにおいて、前記バス・コマンドがリード動作である場合に、
前記第２バス・ブリッジは、前記リード動作に従ってデータを取り込み且つ前記アドレス範囲に従ってデータを先取りし、データは要求されたときに前記バス・マスタに供給される、コンピュータ・システム。
【請求項３３】　請求項３２記載のコンピュータ・システムにおいて、前記バス・マスタが前記アドレス範囲の最後のアドレスにおいて読み取りを行うときに、前記第２バス・ブリッジは前記データ・バッファの予約を取り消す、コンピュータ・システム。
【請求項３４】　請求項２９記載のコンピュータ・システムにおいて、前記予約コマンドは、フレーム信号がアサートされた後に受け取られる、コンピュータ・システム。

【請求項３５】　請求項２９記載のコンピュータ・システムにおいて、前記第２バス・ブリッジおよび前記バス・マスタは周辺素子相互接続（ＰＣＩ）バスに結合される、コンピュータ・システム。

【請求項３６】　請求項２９記載のコンピュータ・システムにおいて、前記バス動作は、前記大容量記憶システムに書き込むためのメモリ・リード動作を含む、コンピュータ・システム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、一方のバスから他方のバスに動作を変換する方法に関し、更に特定すれば、バス間でデータを最適に転送するためのバッファ予約機構(buffer reservation mechanism)に関するものである。

【０００２】

【従来の技術】パーソナル・コンピュータは、新たな技術が発展しコンピュータに組み込まれるに連れて、常に変化しつつある。マイクロプロセッサおよびメモリの性能向上の結果、コンピュータは非常に強力になり、以前では大型メインフレーム・コンピュータによってのみ処理可能であったタスクも、今や処理可能となっている。しかしながら、メインフレーム・コンピュータに完全に取って代わるには、コンピュータは心臓部のＩ／Ｏ（入力／出力）サブシステムが大量のメモリおよび格納容量に対応していなければならない。

【０００３】ＩＳＡ(Industry Standard Architechture:業界標準アーキテクチャ)、ＥＩＳＡ(Extended Industry Stabdard Architecture:拡張業界標準アーキテクチャ）、およびＰＣＩ(Peripheral Component Interface:周辺素子インターフェース)を含むいくつかの標準化されたＩ／Ｏバスが、システム設計者には使用可能である。今日のコンピュータは、典型的に、これら３種類のいずれかを組み合わせることによって設計される。これらバス間でデータを移動するためには、ブリッジ・デバイス(bridge device)が典型的に設けられる。

【０００４】ブリッジ・デバイスは、双方のバスを接続し、これらバス間でデータを転送すると共に、バス制御信号の変換を行う。バスは異なるものとすることができ、あるいは、特にＰＣＩの場合には、ブリッジは単に同一の論理バスに対する電気的な拡張（エクステンション）を与えることができる。この電気的な分離のために、一方のバス・セグメント上における物理的デバイス数を制限しつつ、同時にＰＣＩエージェントの総数は制限しない、ＰＣＩバスの要件を満たすことができる。ＰＣＩバスの用語では、エージェントとは、マスタ・デバイスおよびスレーブ・デバイスを含む、バスに接続する種類のデバイスを示す言葉である。

【０００５】これらのバスは、全て、バス・マスタリング(bus mastering)と呼ばれる方式に対応しており、通常プロセッサ以外のデバイスまたはエージェントが、アービタ(arbiter)にバスの制御を要求することができる。アービタがエージェントに制御を許可した場合、このエージェントはバス・マスタとなる。バス・マスタはその処理を、スレーブと呼ばれる別のエージェントに差し向ける。次いで、バス・マスタは、プロセッサの介入なしに処理を実行することができ、プロセッサを伴う場合よりも格段に効率的である。多くの場合、発生元バス(origination bus)上のマスタは、宛先バス(destination bus)上のスレーブと通信する。これらのバスがブリッジで結合されている場合、ブリッジの性能がマスタとスレーブ間の通信性能に大きく影響する。従って、この経路を最適化することが望ましい。

【０００６】ＰＣＩバスの場合、性能を向上させる方法の１つは、遅延トランザクション又はリード・ポスティング(read posting)を可能にすることである。ＰＣＩバスおよび遅延トランザクションに関する更なる詳細は、オレゴン州ヒルズボロ（Hillsboro）のＰＣＩスペシャル・インタレスト・グループ（Special Interest Group）から入手可能な“ＰＣＩローカル・バス仕様　バージョン２．１（PCI LocalBus Specification version 2.1）”に記載されている。ＰＣＩローカル・バス仕様およびそれに関連する文書をここに参照として援用する。遅延トランザクションは、低速デバイスがリクエスト（要求）に応答してデータを用意している間、バスの使用を可能にする。従って、低速デバイスが待ち時間をバスに加える代わりに、バスを他の要求のために用いることができる。ブリッジにとって、宛先バスは、元の要求のターゲット(target)として動作しなかったインターフェースを意味する。遅延トランザクションは、マスタによる要求、ターゲットによる要求の完了、およびマスタによるトランザクションの完了という３つの段階（フェーズ）で、完了に至る。

【０００７】第１フェーズでは、マスタはバス上にトランザクションを発生し、ターゲットがアクセスをデコードし、アクセスを完了するために必要な情報をラッチし、リトライ－終了(retry-termination)によって要求を終了する。マスタは、遅延トランザクション終了を用いてトランザクションを完了しつつあるターゲットと、現時点において単にトランザクションを完了できないターゲットとの区別ができないので、要求を再度発行しなければならない。第２フェーズでは、ターゲットは、遅延要求からラッチした情報を用いて、宛先バス上で独立して要求を完了する。遅延要求が読み出し（リード(read)）の場合、ターゲットは、要求データおよび完了ステータスを取得する。遅延要求が書き込み（ライト(write)）の場合、ターゲットはライト・データを送り、完了ステータスを取得する。第３フェーズの間、マスタは首尾良くバスの再調停を行い、元の要求を再発行する。ターゲットはこの要求をデコードし、完了ステータス（お

よび、リード要求の場合はデータ）をマスタに供給する。発生元バス上で完了する前に宛先バス上で完了しなければならない全てのバス・コマンドは、遅延トランザクションとして完了させることができる。これらは、割込アクノレッジ（承認）、Ｉ／Ｏリード、Ｉ／Ｏライト、コンフィギュレーション・リード、コンフィギュレーション・ライト、メモリ・リード、メモリ・リード・ラインおよびメモリ・リード多重のコマンドを含む。メモリ・ライト、およびメモリ・ライト及び無効化(invalidate)のコマンドは、宛先バス上で完了する前に、発生元バス上で完了することができる。これらのコマンドは、遅延トランザクション終了を用いて完了されず、通常はポスト(post)される。

【０００８】かかるブリッジ・デバイスの１つが、IntelのＰＣＩ－ＥＩＳＡブリッジ・チップ・セットである。82375EB/SB PCI-EISA Bridgeおよび82374EB/SB EISAシステムのコンポーネントは、対で動作してＰＣＩバスを有するコンピュータにＥＩＳＡ Ｉ／Ｏインターフェースを提供する。チップ・セットは、ＰＣＩバス及びＥＩＳＡバスの両方上のマスタ又はスレーブの何れでもよい。ＰＣＩ－ＥＩＳＡデータ転送では、４つの３２ビット・ポステッド・ライト・バッファ(32-bit posted write buffer)が設けられ、単一サイクルＰＣＩバス・トランザクションの改善を図っている。ＥＩＳＡ－ＰＣＩデータ転送では、４つの１６バイト・ライン・バッファが含まれ、ＥＩＳＡバースト処理(EISA bursting)を支持する。

【０００９】双方のバスを効率的に使用するために、殆どのブリッジは、当該ブリッジ自体に、ある量のデータ・バッファ機能を実施している。これによって、ブリッジはバス同士を互いに結合解除し、他方のバスによる速度低下を招くことなく、各バスをその最大速度で処理させることが可能となる。通常、ブリッジ内に実施可能なバッファには２種類があり、それらは、ライト・ポスティング・バッファ(write posting buffer)およびリード・プリフェッチ（先取り）・バッファ(read prefetch buffer)又はリード・アヘッド・バッファ(read ahead buffer)である。両種類共、いずれのバス上でも実施可能である。

【００１０】ライト・ポスティング・バッファは一方のバスからのライト・データを受け入れ、当該バスに受け取りを知らせる。これによって、バスを解放し他のトランザクションを処理させる。ブリッジは、他のバスに書き込むことができるようになるまで、ライト・データを一時的に格納、又はポスト、する。リード先取りバッファは、単一のリード・アクセスからアドレスを取り込み、追加のデータも必要となることを推測してその読み取りを行う。次いで、ブリッジは、バッファが使用不可能になるかあるいはリード・アクセスによって使用されるまで、このデータをバッファ内に保持する。

【００１１】インテル（Intel）のチップ・セットでは、バッファによって、ブリッジはピーク・データ転送速度で短いデータ・バーストを受け取ることができるようになる。例えば、ＥＩＳＡデバイスがＰＣＩバス上のメモリからのデータを要求すると、ブリッジは、ＰＣＩメモリからそのバッファに４つの３２ビット・データ・ワードをバースト状に入力し、ＥＩＳＡデバイスがバッファを読み取っている間、ＰＣＩバスを他のＰＣＩ要求元に解放することができる。したがって、ＰＣＩバスは、ＥＩＳＡデバイスにより遅延されない。バッファが満杯の間、ＥＩＳＡデバイスは、バッファからデータを読み取るように通知される。こうして、ＥＩＳＡデバイスは、そのバースト転送速度でバッファからデータを読み出すことができる。その後、ブリッジは、ＰＣＩバスを通じて短いデータ・バーストを実行しバッファを満杯に維持することによって、ＥＩＳＡバス転送速度を維持しようとする。しかしながら、他のＥＩＳＡデバイスがメモリからのデータを要求した場合、バッファを空にして（フラッシュして）再度満たさなければならない。

【００１２】また、バッファは、データの一貫性に伴う問題の原因にもなる。データをブリッジ内にバッファしている間、バス・エージェントおよびプロセッサが、メモリ内に実際に何が入っているかについて、異なる考えを有する場合がある。バス・マスタがブリッジを通じて、ブリッジの他方側のターゲットのメモリ範囲にリード要求を発行した場合、ブリッジは、処理能力およびデータ完全性（インテグリティ（integrity））という２つの矛盾する目標の間で、バランスを取らなければならない。処理能力のためにメモリ動作および転送を最適化する場合、ブリッジは、好ましくは、大きなブロックの先取りしたデータを保持するであろう。しかしながら、データを大きなブロックで先取りしたものの、使用されない場合には、ブリッジは、マスタによる後からのリード要求に対して、古いデータを供給することを防止しなければならない。加えて、使用されないデータの大きなブロックを先取りすると、バス帯域を浪費することにより、性能低下を招く。したがって、性能要件(performance concerns)を満たしつつ、データの完全性も保証する解決策を見いだすことが望ましい。

【００１３】バスは異なる周波数でも動作可能である。したがって、ブリッジは、信号をある周波数の一方のバスから別の周波数の他方のバスに変換する際に、信号を同期させなければならない。連続データ・フローを達成すべき場合、同期は性能を犠牲にしてはならない。

【００１４】

【課題を解決するための手段】２つのバスを接続するバス・ブリッジは、複数のデータ・バッファを含む。データ・バッファは、リード・データを格納するため、先取りしたデータ又はリード・アヘッド・データを格納するため、あるいはライト・ポステッド・データを格納する

ために用いられる。本発明によれば、これらデータ・バッファの使用によって、デバイスがバスにアクセスする回数が減少し、バスのレイテンシ（latency）が減少する。

【００１５】ブリッジは、更に、データ・バッファをリザーブ（予約）する機構を含む。２つの代替機構が開示され、これにより、ブリッジに結合されているバス・マスタまたはエージェントは、ブリッジがバス・マスタの意図するデータ・リード転送のためにバッファを予約するように、要求することができる。ブリッジ階層がある場合、予約要求（リザベーション・リクエスト）は、バッファ予約機構を支持するアップストリーム（上流）・ブリッジに渡される。

【００１６】ブリッジが受け取る予約要求は、バス・マスタがデータを予約したデータ・バッファから除去する際に、ブリッジがデータを予め読み取る（リード・アヘッドする）又は先取り（プリフェッチ）するために必要なアドレシング情報を含む。予約が行われる場合、３つの１２８バイト・データ・バッファがバス・マスタのために予約される。データは順番にそして巡回的に、マスタが読み取るに連れて、これらのデータ・バッファに先取りされる。データ・バッファが最初にマスタによってアクセスされるときに、次のデータ・バッファが先取りされる。アドレス範囲全体が読み取られ、データがバス・マスタによって除去されたときに、予約は取り消される。プログラム可能な時間期間内に、バッファがバス・マスタによってアクセスされない場合も、予約は取り消される。予約バッファを管理するための予約アルゴリズを開示する。

【００１７】ブリッジは、更に、予約機構がエージェントに使用可能でない場合に、バッファを管理する２つの代替アルゴリズムも含む。デフォルト・アルゴリズムを用いると、新しいバス・マスタが最初にバッファを予約せずにリードを行おうとした場合、バッファが使用可能であれば、ブリッジが３つの１２８バイト・データ・バッファから成るバッファをこのマスタに割り当てる。その他の場合、マスタは、リード動作をリトライするように強制される。一旦バッファが割り当てられたなら、ブリッジは要求されたデータを１２８バイト・データ・バッファの最初のものに読み込み、その後データをバス・マスタに供給する。マスタが、第１データ・バッファの終端を越えてバースト・リードを行い、マスタからの次のリード動作が、最後のリードが終了した位置から開始される場合、このマスタのために先取りがイネーブルされる。次のシーケンシャルのリードの時に、ブリッジは再び要求されたデータを第２データ・バッファに読み込み、その後それをマスタに供給する。この時点では、先取りは既にイネーブルされている。マスタが、バッファの１つの終端を越えて再度バースト・リードを行うときはいつでも、ブリッジは自動的に次の１２８バイト・バッファに対してデータを先取りする。３つの１２８バイト・データ・バッファは、順番にそして巡回的に満たされる。バス・マスタが不連続なリードを要求した場合、またはＣＰＵがマスタに影響を与え得るライト・トランザクションを実行した場合、バッファはフラッシュされる。

【００１８】保存アルゴリズムは、デフォルト・アルゴリズムと同様であるが、ブリッジは先取りを行わない。バス・マスタがデータを全て読み取らない場合、マスタがバスからディスコネクト（接続解除）されるときに、残りのデータをフラッシュする。

【００１９】

【発明の実施の形態】次に図１を参照すると、好適実施例によるブリッジ・デバイス１０６が図示されている。ブリッジ・デバイス１０６は、コンピュータ・システム１００の一次バス１１０を二次バス１１２に結合する。コンピュータ・システム１００は、更に、一次バス１１０に結合されているＣＰＵ１０２およびメモリ１０４を含む。好適実施例では、ＣＰＵ１０２は、ＩＢＭコンパチブルのＰＣにおいて共通するような、インテルの４８６，ペンティアム７（Pentium7）または５８６級のプロセッサまたはそれらの同等物のようなプロセッサ（図示せず）、ならびに標準的なコンピュータ周辺装置および支援ロジック（図示せず）を含む。他のプロセッサや周辺素子も勿論使用可能であることは理解されよう。

【００２０】ハード・ディスク・コントローラ、ビデオ・グラフィック・コントローラまたはネットワーク・インターフェース・コントローラのようなバス・マスタ１０８、およびシリアル（直列）・ポートまたはパラレル（並列）・ポートのようなバス・スレーブ１１４が、更に、二次バス１１２に接続されている。本発明において開示される原理の目的のためには、バス・マスタ１０８およびスレーブ１１４の具体的な機能性を示すことは不要であるので、ここでは、単に一般的なバス・マスタおよびスレーブとして言及するに止めておく。

【００２１】一次バス１１０および二次バス１１２は、周辺素子相互接続（ＰＣＩ）バスのような、標準的なＩ／Ｏバスである。尚、ここに開示する原理は、業界標準アーキテクチャ（ＩＳＡ）、拡張業界標準アーキテクチャ（ＥＩＳＡ）およびマイクロチャネル・アーキテクチャ（ＭＣＡ：microchannel architecture）バスのような他のバスにも適用可能であることは理解されよう。更に、同じブリッジ設計原理は、ＰＣＩバスおよびＥＩＳＡバスのような２種類の異なるバスを結合するために適用可能であることも、認めることができよう。好適実施例では、一次バスは、電気的ファンアウト(electrical fanout)を限定したローカルＰＣＩバスであり、ブリッジ１０６は、２次のＰＣＩバス１１２によって一次バス１１０の機能性を拡張するために用いられる。

【００２２】図２に示す図１の代替構成では、二次バス

１１２は、コンピュータ・システム１００から離れた場所にある。更に、ブリッジ１０６は、アップストリーム（上流）部１０６ａおよびダウンストリーム（下流）部１０６ｂを備える。ケーブル１１８がアップストリームとダウンストリームのブリッジ・インスタンスを互いに結合する。ダウンストリーム部１０６ｂは二次バス１１２に結合され、複数のバス・マスタ１０８ａないし１０８ｄとの通信を行う。各バス・マスタは、コンピュータ・システム１００に大容量記憶装置を提供するために、複数のハード・ディスク・ドライブ１１６ａないし１１６ｄにも結合される。図１および図２の双方において、ブリッジ１０６は、バス・マスタ１０８のような二次バス上のデバイスに共通インターフェースを与え、メモリ１０４のような一次バス上のデバイスとの通信を可能にする。また、バス・マスタ１０８は、一次バス１１０からの帯域を全く要求することなく、バス・スレーブ１１４のような、その他の二次バス・デバイスとも通信することができる。したがって、変換ユニットおよびバス拡張部であることに加えて、ブリッジ１０６は分離バッファ(isolation buffer)としても機能する。

【００２３】バス・マスタ１０８がメモリ１０４に対して読み取りまたは書き込みを行う場合のように、大きなデータ・ブロックをブリッジを通じて転送する際、ブリッジは、リード先取り(read prefetching)およびライト・ポスティング(write posting)のための多数のバッファを提供し、全体的なシステム性能向上を図る。これは、転送が発生し得る前に、双方のバスがブリッジ１０６と通信するという、典型的な要件をなくすのに役立つ。リード先取りの管理は、いくつかの代替的なアルゴリズムによって行われる。アルゴリズムの１つは、バス・マスタが非常に効率的なバス転送のためにリード先取りバッファを予約可能とする機能を含む。他の適応的アルゴリズムは、バス・マスタがバッファ予約を要求しない場合を扱う。しかしながら、これらの場合でも、リード先取りが高い成功確率を有することが所定の状態によって示された場合には、アルゴリズムはバッファを予約しデータを先取りしようとする。

【００２４】次に図３および図４に移ると、ブリッジ・デバイス１０６の第１および第２実施例のブロック図が示されている。図に見られるように、一次バスは一次バス・インターフェース２００に結合し、二次バスは二次バス・インターフェース２０２に結合する。ブリッジ１０６は、入来データを受け取り、リード先取り又はリード・アヘッドを行い、また、ライト・ポスティングを行うためのバッファ領域２０４を含む。バッファは、処理の種類に応じたコンフィギュレーションに設定される。これについては、以下で説明する。

【００２５】具体的には、ブリッジ１０６には、ライト動作の処理を向上させるために、４つの１２８バイト・ライト・ポスティング・バッファ２１８が設けられる。バス・マスタ１０８が主メモリ１０４に対してライト動作を実行する場合、データはライト・バッファ２１８に書き込み又はポスト(post)され、バス・マスタ１０８は、データが実際に主メモリ１０４に書き込まれる前に、ブリッジ１０６から完了の指示（indication）を受け取る。こうして、ブリッジ１０６は後から主メモリ１０４に対するライト動作を完了し、遅延を生じない。このように、実際の完了指示を待たないので、バス・マスタ１０８が二次バス１０２を制御する時間は大幅に短縮される。

【００２６】また、バッファ領域２０４は、主メモリ１０４からのバス・マスタ・リード動作を向上するために、８つのデフォルト・バッファ２１０も含む。デフォルト・バッファ管理アルゴリズムがデフォルト・バッファ２１０を管理し、所定の条件の下において、リード・データの格納およびリード・データの先取りを行う。デフォルト・バッファ管理アルゴリズムは、できるだけ長く先取りデータを保持しながらも、古いデータを残す可能性を回避するという２つの間のバランスを取る。バッファ内の３２ビット・ダブル・ワードのデータのチャンク(chunk)（以下DWORDSと呼ぶ）に対する３つの３ビット・ポインタを用いて、バッファを満杯にしたり空にしたりする。バッファ入力ポインタ(BUFF_IN_PNTR)は、次のDwordが格納される場所を指し示す。バッファ出力ポインタ(BUFF_OUT_PNTR)は、次のDwordが読み出される場所を指し示す。バッファ有効ポインタ(BUFF_VALID_PNTR)は、BUFF_IN_PNTRとBUFF_OUT_PNTRの間のどちらのアドレスが有効であるかを示す。バスがロックされる状況を防止するために、ポストされたメモリ・ライト(posted memory write)の後に完了したリードは、バッファに格納することができるが、ポストされたメモリ・ライトが完了する後までは有効となることができないので、所定のアドレスのみが有効となるようにすればよい。

【００２７】デフォルト・バッファ管理アルゴリズムの代わりとして、デフォルト・バッファ２１０を管理するために保存アルゴリズムが用意されている。保存アルゴリズムは先取りを行うことはなく、通常はバス・マスタによって要求されたデータを読み取るのみである。したがって、古いデータは完全に回避される。

【００２８】更に、バッファ領域２０４は、主メモリ１０４からのバス・マスタ・リード動作の向上を図るために、８つの予約リソース（資源）２２０のプールも含む。これらの資源２２０は、以下で論ずる予約アルゴリズムによって管理される。各予約資源２０６は、３つの１２８バイト・データ・バッファと、対応する予約情報ブロック２０８で構成されている。各情報ブロック２０８は、予約バッファを制御するのに必要なレジスタを含む。かかるレジスタには、６４ビットの予約の開始アドレスを収容する予約ベース・アドレス・レジスタ(RSRV_BASE_ADDR)、予約の最後のアドレスよりも１大きな６４

ビット・アドレスを収容する予約制限アドレス・レジスタ(RSRV_LIMIT_ADDR)、非アクティブな予約資源を自動的に取り消すために４つの値（オフ、２５ｍｓ、５０ｍｓ、または１００ｍｓ）の１つにプログラムされた破棄タイマ値、データが読み込まれた最後のライン・バッファへの５ビット・ポインタを収容する最後バッファ・インディケータ・レジスタ(LAST_BUFF)、各ライン・バッファの６４ビット開始アドレスを収容するバッファ開始ポインタ・レジスタ（BUFF_START_PNTR)、ライン・バッファの最終アドレスよりも１大きな６４ビット・アドレスを収容するバッファ制限ポインタ・レジスタ(BUFF_LIMIT_PNTR)、次のDwordが格納されるアドレスを保持するバッファ入力ポインタ・レジスタ(BUFF_IN_PNTR)、次のDWORDが読み出されるアドレスを保持するバッファ出力ポインタ・レジスタ(BUFF_OUT_PNTR)、BUFF_IN_PNTRとBUFF_OUT_PNTRの間でどちらのアドレスが有効かを示すバッファ有効ポインタ・レジスタ(BUFF_VALID_PNTR)、バッファ状態レジスタ、およびバッファ先取りフラグ・レジスタが含まれる。

【００２９】デフォルト・バッファ２１０および予約資源２２０は、各要求元マスタあるいは各プロセスまたはスレッドに割り当てられてもよく、こうすることにより、単一のマスタが多数のバッファを有することが可能となる。バッファ領域２０４はＦＩＦＯとして編成されており、勿論、最適なバス利用のために、マスタの要件にしたがってサイズを決定することができる。ブリッジ１０６は要求元およびそのターゲットには透明であることが好ましいので、バッファ領域２０４もトランザクションに対して透明であることが望ましい。しかしながら、バッファ領域２０４の動作を最適化するには、データが効率的に先取りされ、先取りされたデータが主メモリ内のデータとコヒーレントとなるようにバッファを制御することが望ましい。したがって、未処理のトランザクション(pending transaction)に対してバッファを最適化するには、マスタが読もうとしているアドレス範囲をブリッジが受け取ることが望ましい。こうすれば、一旦ブリッジがアドレス情報を取得したなら、ブリッジはアドレス範囲の最後まで自由にデータを先取りすることができる。予約資源を要求するための２つの代替案が用意されている。

【００３０】図３に示す第１の代替的な予約機構において、二次インターフェース２０２はコマンド・デコード・ブロック２１２ａを含む。メモリ・リード要求は下流側のバス・マスタからなので、コマンド・デコード・ブロック２１２ａは二次インターフェース２０２に配置されているが、逆方向のリード動作では、コマンド・デコード・ブロックが一次インターフェース２００においても有効であることに留意されたい。予約資源は、主メモリ１０４から、二次バス１１２上に位置するバス・マスタから開始されるリード動作に対して最も効果的であるので、以下の説明はこの文脈に沿って進めることにする。

【００３１】予約資源２２０は、特殊なバス・コマンドがブリッジ１０６に受け取られた場合に、バス・マスタ１０８によって予約することができる。この点について、読者は図７も参照されたい。バス・マスタ１０８がターゲット・メモリ１０４からのリード・トランザクションのために予約資源２２０を予約したい場合、バス・マスタ１０８は、ターゲット・メモリ１０４から読み出される合計バイト数を示す予約コマンド(RSRV_COMMAND)を含む、第１リード・トランザクションを発行する。RSRV_COMMANDは、ＰＣＩバス上のコマンド／バイト・イネーブル(C/BE#)信号によって表される特殊なＰＣＩバス・コマンドであり、図７のクロックの２に示すように、フレーム(FRAME#)信号がアサートされトランザクションの開始が示されたときに、第１リード・トランザクション上でブリッジ１０６に提示される。また、予約コマンドは、AD[31:0]ライン上のカウントも含み、所望されるバイト数を示す。クロックの３以降では、ＰＣＩトランザクションは通常通りに進む。即ち、ＰＣＩトランザクションの種類が、アドレス／データ(AD)ライン上の開始アドレスと共に、C/BE#ラインに供給され、入手可能なときにデータが転送される（クロックの５，７および９に示す通り）。したがって、カウント（クロックの２）および開始アドレス（クロックの３）は、読み取るべきアドレス範囲を示す。このバッファを予約する範囲を示すには、開始アドレスおよび終了アドレスを示すことによる等のように、代替方法もあることは理解されよう。これらの代替方法は、予約コマンドのために考えられたものである。

【００３２】図８に示す６４ビットのアドレシングでは、RSRV_COMMANDがデュアル（二重）・アドレス・コマンドおよび実際のリード・コマンドの前に位置する。二重アドレス・コマンドについての更なる詳細は、先に引用したＰＣＩ仕様に記載されている。

【００３３】ブリッジ１０６がRSVR_COMMANDを受け取った場合、予約情報ブロック２０８のレジスタが初期設定される。開始アドレスはRSRV_BASE_ADDRに格納され、終了アドレスは、開始アドレスに合計バイト・カウントを加算することによって決定され、１だけ増分されてRSRV_LIMIT_ADDRに格納される。３つの１２８バイト・バッファの内１つが、開始アドレスをBUFF_START_PNTRに格納し、バッファ終了アドレスをBUFF_LIMIT_PNTRに格納することによって、初期設定される。また、３つのバッファ各々について、バッファ状態および先取りフラグも初期設定される。

【００３４】一旦予約が行われたなら、バス・マスタ１０８はブリッジ１０６からディスコネクトし、後に再接続することにより、RSVR_COMMANDを再発行することなく処理を再開することができる。したがって、この代替案

では、単一のＰＣＩバス・コマンドが、転送すべきバイト数を示す。一旦バッファが予約されたなら、データ・トランザクションは正常に進行するので、この追加情報に対する負担は１クロック・サイクルのみである。

【００３５】ターゲットはRSRV_COMMANDを受け取らない。しかしながら、中間ブリッジが、下流（ダウンストリーム）側ブリッジからのRSRV_COMMANDを受け取る場合がある。図５および図６に示すように、ブリッジの階層構造がある場合、RSRV_COMMANDを支持する各ブリッジは、予約を上流（アップストリーム）に渡すことができる。各ブリッジのコンフィギュレーションでは、上流側ブリッジがバッファ予約を支持していない場合、コンフィギュレーション・ビット２１４（図３）がクリアされ、その他の場合、このビットはコンピュータの初期設定処理の間にコンフィギュレーション・ソフトウエアによってセットされ、予約（リザベーション）は上流に渡される。図５は、ブリッジ階層の一例を示し、この場合、ブリッジ３００は、バッファ予約に対応する（バッファ予約を支持する）２つのブリッジ１０６ａと１０６ｂの間に挟まれた非予約ブリッジである。この例では、ブリッジ３００がバッファ予約に対応していないので、ブリッジ１０６ｂ内のコンフィギュレーション・ビットはクリアされ、RSRV_COMMANDは非予約ブリッジ３００には送出されない。したがって、ブリッジ１０６ａはバッファ予約に対応しているが、この特徴はブリッジ１０６ａでは利用されない。

【００３６】図６はブリッジ階層の一例を示し、この場合、最上位のブリッジ３００はバッファ予約に対応していないが、２つの下位ブリッジ１０６ａおよび１０６ｂはそれに対応している。この例では、ブリッジ３００はバッファ予約に対応していない（バッファ予約を支持しない）ので、非予約ブリッジに最も近いブリッジ、即ち、ブリッジ１０６ａ内のコンフィギュレーション・ビットをクリアし、RSVR_COMMANDが非予約ブリッジ３００に送出されないようにする。ブリッジ１０６ｂ内のコンフィギュレーション・ビット２１４はセットされ、ブリッジ１０６ｂによって受け取られたRSRV_COMMANDがブリッジ１０６ａに送出されるようにする。このようにして、非予約ブリッジは、RSRV_COMMANDが実施される場合に、バッファ予約の効果を切り離すように作用する。

【００３７】次に、図４に示す第２の代替的な予約機構を参照すると、ブリッジ１０６は、特別なアドレスに対する処理をデコードするためのアドレス・デコード・ブロック２１２ｂを含む。コンピュータ・システム１００のメモリ・アドレス・マップでは、この特別なメモリ・アドレスは、この予約機構のために割り当てられている。この特別アドレスに対するライト処理は、ブリッジ１０６による予約動作として解釈される。バス・マスタ１０８がリード動作のためにバッファ予約を望む場合、まず、２－３　３２ビット(Dword)バースト・ライトを特別アドレスに実行する。ブリッジ１０６はこのライト動作を横取り、コマンドを予約メールボックス(reservation mailbox)２１６に格納する。最初のDwordは、バス・マスタ１０８が読み取ろうとする合計バイト数である。２番目のDwordは開始アドレスの下位３２ビットである。６４ビット・アドレシングを用いる場合、開始メモリ・アドレスの上位３２ビットを収容する第３のDwordが書き込まれる。２つのDwordのみが書き込まれる場合、ブリッジは上位３２ビットがゼロであると仮定する。ブリッジ１０６はデバイス・セレクト(DEVSEL#)信号をアサートすることによって、メールボックス・レジスタ２１６に対するライト処理を横取りし要求する。何故なら、アドレスは二次バス１１２上のいずれのデバイスにも割り当てられていないからである。ブリッジ１０６がトランザクションを受け取るとき、バッファが使用可能であればそれを予約し、必要であれば、他の中間ブリッジに送出するために、メールボックス２１６にアドレス、長さおよびその他の情報を格納する。予約が行われた場合、上述のように、予約情報ブロックの初期設定も行う。したがって、この代替案では、特別アドレスに対するライトが、開始アドレスおよび転送すべきバイト数を示す。この情報は、ブリッジ１０６が要求元マスタ１０８のためにデータを先取りするために利用する。

【００３８】図５および図６に示すように、ブリッジ階層がある場合、バッファ予約を支持する各ブリッジは、予約情報を上流に渡すことができる。各ブリッジのコンフィギュレーションでは、階層内の最上位のブリッジは、予約を送出することをディスエーブルにされる。図５は、バッファ予約を支持する２つのブリッジ１０６ａと１０６ｂの間に挟まれたブリッジ３００が非予約ブリッジであるブリッジ階層を示す。この例では、グローバル・メモリ・アドレスに対するライト・トランザクションを用いてブリッジ間で予約情報を受け渡すので、コンフィギュレーション・ビットおよびブリッジ１０６ｂがセットされ、予約が非予約ブリッジ３００に送出され、予約ブリッジ１０６ｂに渡されるようにする。非予約ブリッジ３００は、通常のメモリ・トランザクションと予約ライト・トランザクションとの間の相違を認識しない。ブリッジ１０６ａがその一次側でメモリ１０４に結合されている場合、コンフィギュレーション・ビットおよびブリッジ１０６ａが、メモリへの予約情報の送出をディスエーブルするようにセットされる。このようにして、バッファ予約に対応可能な各ブリッジは、予約情報を受け取る。

【００３９】図６は、最上部のブリッジ３００はバッファ予約に対応しないが、下位２つのブリッジ１０６ａおよび１０６ｂはそれに対応する、ブリッジ階層を示す。この例では、ブリッジ３００は最上位ブリッジであり、バッファ予約を支持しないので、ブリッジ１０６ａのコンフィギュレーション・ビットはクリアされ、予約情報

の送出がディスエーブルされる。予約はブリッジ１０６ａと１０６ｂの間で受け渡しされる。このように、非予約ブリッジが最上位に位置する場合、非予約ブリッジの下にある予約ブリッジは、予約の送出をディスエーブルされる。最初の代替案に対しての、この方法の利点の１つは、スクリプト・ドリブン・バス・マスタ(script-driven bus master、スクリプト駆動のバス・マスタ)は、殆ど変更せずに、メモリ・ライト・トランザクションを容易に実施できることである。

【００４０】双方の代替案は、より長くより効率的なトランザクションにおいて、ブリッジ１０６がデータを先取り（プリフェッチ）できるようにするものである。バッファ予約を利用することにより、要求されたデータのみが先取りされるので、使用されないデータが先取りされることが回避される。要求されたデータのみが先取りされ、要求元が転送を完了するまで、バス・マスタはこのデータを使用しないと考えることができるので、メモリ１０４とバッファ２０６および２１０に格納されているデータの間のコヒーレンシに関する問題も回避される。このように、好適実施例にしたがって先取りされたデータは、転送が完了するまで保持され且つ有効と見なすことができる。

【００４１】遅延トランザクション（delayed transaction）として知られている技法によって、リード動作をブリッジ１０６によって実行することができる。遅延トランザクションに関する更なる詳細は、上記で援用したＰＣＩ仕様及びこの明細書の従来技術の項に記載されている。しかしながら、端的に言えば、遅延トランザクションとは、所定の指定時間内にデータ・トランザクションを完了することができないＰＣＩブリッジおよびＰＣＩ　Ｉ／Ｏによって用いられる、終了およびリトライ（retry、再試行）の方法である。第１の要求元バス・マスタが低速のＩ／Ｏコントローラまたはブリッジが応答するのを待つ代わりに、このＩ／Ｏコントローラまたはブリッジは第１の要求元バス・マスタをバスからオフに強制し、待ち状態の第１のバス・マスタが保持することによって通常は浪費されるバス帯域を、他のバス・マスタに使用させる。通常、遅延トランザクションは、バス・マスタによる要求、ターゲットによる要求の完了、およびバス・マスタによるトランザクションの完了、という３段階で完了するように進展する。

【００４２】ＰＣＩバスの用語では、遅延リード要求とは、発生元のバス上で完了する前に宛先バス上で完了しなければならないトランザクションのことであり、Ｉ／Ｏリード、コンフィギュレーション・リード、メモリ・リード、メモリ・リード・ライン、またはメモリ・リード多重コマンド(memory read multiple command)がそれに該当する可能性がある。図１を参照して、バス・マスタ１０８がメモリ１０４からのデータを要求する場合、二次バス１１２が発生元バスとなり、一次バス１１０が宛先バスとなる。一旦宛先バス上で要求が試行されると、これが宛先バス上で完了するまで、連続して繰り返されなければならない。そのときまで、遅延リード要求は単なる１要求に過ぎず、バス・マスタ１０８はこの要求を後に繰り返さなければならないので、バスのデッドロックを防止するためまたは処理を促進するために、いつでも破棄され得る。遅延リード完了とは、宛先バス上で完了し、完結するために要求元バスに移動しつつあるリード・トランザクションのことである。遅延リード完了は、バス・マスタ１０８によって要求されたデータ、およびターゲット（メモリ１０４）のステータスを含む。一旦遅延リード要求が宛先バス上で実行されたなら、これは遅延リード完了となる。

【００４３】バッファ予約機構を用いない単純な遅延トランザクションの例は、バッファ領域２０４上におけるその効果を示すのに役立つであろう。尚、遅延トランザクションは、ここに開示するバッファ予約機構を用いても、または用いなくても使用可能であることに留意されたい。また、バッファ予約機構は、遅延トランザクションを用いても、または用いなくても使用可能であるが、最大の処理性能を得るために、これらを一緒に用いるものと仮定する。

【００４４】図１を一例として用いる。第１フェーズの間、バス・マスタ１０８は、二次バス１１２上にメモリ・リード・トランザクションを発生する。ブリッジ１０６は、このトランザクションをデコードし、主メモリが一次バス上にあることを認識し、DEVSEL#信号をアサートすることによってトランザクションを要求し、アクセスを完了するために必要なトランザクション情報をラッチし、従来のＰＣＩリトライ技法を用いて要求を終了する。ラッチされた要求情報のことを、遅延要求、あるいは本例では更に具体的に、遅延リード要求と呼ぶ。ＰＣＩの規則によれば、バス・マスタ１０８はトランザクションを繰り返さなければならない。その間、ラッチされたトランザクション情報を用いて、ブリッジ１０６は、一次バス１１０上でメモリ・リード・トランザクションを発生する。ターゲット（主メモリ１０４）の準備ができていない場合、ブリッジ１０６にリトライを発行することにより、他の遅延トランザクションを強制することも可能である。しかしながら、メモリ１０４がリトライを発行しないと仮定すれば、メモリ１０４は要求を処理し、ブリッジ１０６は完了ステータスおよび要求されたデータを、そのバッファ２０４の内の１つに受け取る。遅延要求がライト動作である場合、ブリッジ１０６は主メモリ１０４から完了ステータスを取得する。宛先バス（一次バス１１０）上で遅延要求（遅延された要求）を完了した結果、遅延完了(delayed completion)が生成される。遅延完了は、遅延要求および完了ステータスのラッチされた情報ならびにデータから成る。ブリッジ１０６は、バス・マスタ１０８が最初の要求を繰り返すま

で、データおよび完了ステータスをバッファ領域２０４に格納する。

【００４５】第３フェーズの間、バス・マスタは首尾良くバスの再調停を行い、元のメモリ・リード要求を再発行する。ブリッジ１０６はこの要求をデコードし、処理を要求し、データをバス・マスタ１０８に供給する。この時点で、遅延完了は引っ込められ、トランザクションは完了したことになる。バス・マスタ１０８に返されるステータスは、ブリッジ１０６が、ターゲット（メモリ１０４）から、リード・トランザクションを実行したときに取得したものと正確に同一であり、即ち、マスタ・アボート、ターゲット・アボート、パリティ・エラー、ノーマル、またはディスコネクトである。

【００４６】遅延トランザクション技法を、ここで開示するバッファ予約機構と組み合わせて用いることにより、バス・マスタとバス・スレーブの間のリード動作は、以前よりも効率的に実行可能となる。

【００４７】バッファ領域２０４の管理は、効率的なデータ転送を得るために望ましいものである。バス領域２０４の基本的な構成要素は、１２８バイト・バッファであり、図９のＡでは、バッファ５００と総称的に呼ばれている。リード先取りデータおよびポステッド・ライト・データ(posted write data)は双方ともブリッジ１０６（の異なるバッファ）に格納することができるが、リード先取りデータは、ポステッド・メモリ・ライトが完了するまで、有効となることはできない。データがバッファ５００に書き込まれると、BUFF_IN_PNTRが増分され、次のアドレスを示す。データは、BUFF_OUT_PNTRによって示されるアドレスに始まって、バッファ５００から消去される。BUFF_VALID_PNTRは、バッファ５００のどの領域が有効かを示す。BUFF_OUT_PNTRとBUFF_VALID_PNTRの間のアドレスが有効アドレスとなる。遅延リード要求が開始されると、BUFF_IN_PNTRおよびBUFF_OUT_PNTRは、ＰＣＩトランザクション・アドレスのビット４－２に初期設定される。データの各Dwordが到達すると、BUFF_IN_PNTRによって示されるアドレスに格納され、ポインタが増分される。宛先バス上のターゲットに宛てた、未だ実行されていない以前のポステッド・メモリ・ライトがない場合、BUFF_IN_PNTRを増分するときにBUFF_VALIDポインタを増分することができる。前のポステッド・メモリ・ライトがある場合、このライトを完了した後にBUFF_VALID_PNTRを更新する。バス・マスタ１０８がバッファからデータを取り込む際、各Dwordが除去された後にBUFF_OUT_PNTRが増分される。

【００４８】図９のＢは、予約資源バッファ２０６がどのように構成されているかを示す。予約バッファ２０６は、RSRV_BASE_ADDRによって示される開始アドレス、およびRSRV_LIMIT_ADDRによって示される終了アドレスを有する。予約バッファ２０６は、３つの１２８バイト・データ・バッファ（図９のＡに示した形式の）で構成されている。リード・トランザクションが進展するに連れて、これら３つのバッファが再使用され、RSRV_LIMIT_ADDRからRSRV_BASE_ADDRを減じた長さの１つの長いバッファをシミュレートする。個々のバッファは、各々、BUFF_STARTによって示される開始アドレス、およびBUFF_LIMITによって示される終了アドレスを有する。

【００４９】バッファの状態は、２つの変数、BUFF_STATEおよびPREFETCH_FLAGによって示される。PREFETCH_FLAGは、先取り動作と実際のリード要求との間で区別するために使用される。バッファに格納されているデータの状態を図１０に示す。バッファの状態は、システム・リセット、バッファ・タイムアウト、またはバッファ・フラッシュ(buffer flush)の後にはEMPTY（空）と表される。バッファがEMPTYと示されている場合、それを割り当てに使用可能である。バッファがリード要求に使用されている場合、またはリード先取りを実行している場合、このバッファはREQUEST（要求）と表される。一旦あるデータがバッファに到達し始めたなら、状態はREQUESTからPART_COMP（部分完了）に遷移する。一旦全てのデータが受け取られたなら、バッファの状態はPART_COMPからCOMPLETE（完了）に遷移する。バッファの状態がREQUST、PART_COMPまたはCOMPLETEであり、バッファがフラッシュ（消去、flush)されている場合、状態はEMPTYに戻る。バッファがCOMPLETEと示され、バス・マスタがバッファからのデータ全てを読み取る場合、状態はCOMPLETEからEMPTYに変化する。

【００５０】予約アルゴリズムでは、バッファがフラッシュされるのは、データの最後のDWORDがバッファから除去されるときである。しかしながら、先取りされたデータのフラッシュは、デフォルト・アルゴリズムまたは保存アルゴリズム(conservative algorithm)では適宜行われ、バス・マスタには決して古いデータを与えないことを保証する。このために、バッファの状態を追跡するのは重要である。

【００５１】先取りデータとは、マスタからの実際の遅延リード要求で満たされていたバッファに残されているデータのことを意味するが、マスタが再接続したときには、マスタがディスコネクトする前にデータの一部のみが取り出されている。マスタの先取りデータは、以下の状態の下でフラッシュされる。（１）マスタが、当該マスタの最後のリードとは連続しないアドレスからリードをしようとする場合。（２）ＣＰＵ１０２がこのマスタに影響を与え得るライト・トランザクション、例えば、二次バス・ターゲットへのＩ／Ｏライト、汎用二次メモリ範囲レジスタへのメモリ・ライト、またはこのマスタのためのスロット特定メモリ範囲レジスタへのメモリ・ライトのようなライト・トランザクションを行う場合。（３）二次バス・マスタがこのマスタに影響を与え得るライト・トランザクション、例えば、二次バス・ターゲットへのＩ／Ｏライト、汎用二次メモリ範囲レジスタへ

のメモリ・ライト、このマスタのためのスロット特定メモリ範囲レジスタへのメモリ・ライト、またはいずれかのコンフィギュレーション・レジスタへのＣＰＵ１０２のライトのようなライト・トランザクションを行う場合。

【００５２】PREFETCH_FLAGに対する状態について、図１１を参照しながら説明する。BUFF_STATEがREQUEST、PART_COMPまたはCOMPLETEと示された場合、PREFETCH_FLAGは、リードが先取りか又は実際のREQUESTかを示す。PREFETCH_FLAGがセットされている場合、先取り動作が進行中であり、一方PREFETCH_FLAGがクリアされている場合、実際のREQUESTが進行中である。セット状態からクリア状態への遷移は、以下の式で示される。

【００５３】

A＝（先取り） OR
（マスタ・リード後 AND BUFF_STATE=(PART_COMP OR COMPLETE));
B＝（フラッシュ AND BUFF_STATE=(REQUEST,PART_COMP OR COMPLETE));
C＝（マスタ・リード・トランザクション AND 遅延リード要求）OR
（先取り完了 AND BUFF_STATE=EMPTY) OR
（フラッシュ AND BUFF_STATE=COMPLETE) OR
（実際の遅延リード要求）;
D＝（マスタ・リード・トランザクション AND 遅延リード要求）OR
（先取り完了 AND BUFF_STATE=EMPTY);

【００５４】バッファ２０４は、それがEMPTYの場合、またはそれが先取りデータを収容しているが当該バッファに対して進行中の先取りが既にない場合にのみ、新しいREQUESTに対して使用可能となる。バス・マスタ１０８がリードを行おうとして、ブリッジ１０６が遅延リード要求を開始した場合、BUFF_STATEはREQUESTに変更され、PREFETCH_FLAGはクリアされる。ブリッジ１０６の先取りによってリード動作が開始された場合、BUFF_STATEはREQUESTに変更され、PREFETCH_FLAG がセットされる。

【００５５】最初のデータが到達したときに、BUFF_STATEはPART_COMPに変更される。マスタがリード・トランザクションを繰り返し、状態がPART_COMPまたはCOMPLETEである場合、リード・データをバス・マスタ１０８に供給する。バス・マスタ１０８が、ブリッジ１０６からディスコネクトする前に、全てのデータを取り込まなかった場合、PREFETCH_FLAGがセットされ、バッファ２０４内のデータの残りはこの時点では先取りされたデータと見なされることを示すが、状態は不変のままである。バス・マスタ１０８が最後のデータを取り込んだ場合、即ち、状態がCOMPLETEで(BUFF_OUT_PNTR >;= BUFF_IN_PNTR >; リード後）の場合、状態はEMPTYに変更される。

【００５６】バッファ２０４に対してフラッシュ・イベントが発生し、そのPREFETCH_FLAGがセットされ、状態がREQUEST、PART_COMPまたはEMPTYと示されている場合、先取りが現在進行中であるので、状態はEMPTYに変更されるが、未処理の先取りが完了するまで、PREFETCH_FLAGはセットされたままである。バッファ２０４は、PREFETCH_FLAGがクリアされるまで、他の先取りのために再使用することはできない。バッファ２０４に対してフラッシュ・イベントが発生し、そのPREFETCH_FLAGがセットされ、状態がCOMPLETEと示された場合、状態はEMPTYに変更され、PREFETCH_FLAGはクリアされる。

【００５７】PREFETCH_FLAGがセットされている間に、実際の遅延要求がバッファ２０４を必要とする場合、新しい要求がバッファに与えられる。PREFETCH_FLAGはクリアされ（入来する先取りデータを破棄させる）、状態はREQUESTに変更される。

【００５８】先取りデータがバッファ２０４に到達し、PREFETCH_FLAGがクリアされると、データは破棄される（バッファは実際の要求のために再使用されている）。先取りデータがバッファ２０４のために到達し、状態がEMPTYの場合、データは破棄される（バッファがフラッシュされる）。最後の先取りデータが到達したとき、PREFETCH_FLAGがクリアされ、バッファ２０４は使用可能となり再使用される。

【００５９】ブリッジ１０６には、３つの先取りおよびフラッシュ・アルゴリズム、即ち、デフォルト・アルゴリズム、保存アルゴリズム、および予約アルゴリズムが実施されている。デフォルト・アルゴリズムは、殆どのバス・マスタに適用可能でなければならない。保存アルゴリズムは、先取りを少なく、フラッシュを多くするが、デフォルト・アルゴリズムが特定のバス・マスタに対して作用しない場合にのみ実施される。予約アルゴリズムは、予約機構を開始可能な、一層高度なバス・マスタによって用いられる。

【００６０】次に図１２ないし図１９を参照しながら、好適実施例による先取り（プリフェッチ）およびフラッシュ・アルゴリズムについて説明する。ブリッジ１０６は、データのリードおよびライトに関する種々の処理を実行するが、簡略化のために、これらのステップを省略してあることは理解されたい。更に、ライト・ポスティングに関連するステップも示されていないが、ブリッジは、ここに記載するリード先取りおよびフラッシュ・プロセスと共に、これらの処理も実行することは理解されよう。

【００６１】先取りおよびフラッシュ・アルゴリズムは、図１２において、二次バス・インターフェース２０

２上のバス・トランザクションの受け取りから開始する。ステップ７００（図１２）において、コンフィギュレーション・コマンドが受け取られたか否かについて判定が行われる。受け取られた場合、制御はステップ７１０（図１３）に移行し、このコマンドを完了する。その他の場合、制御はステップ７０２に進む。ステップ７０２において、ライト動作がブリッジ１０６に発行されたか否かについて判定が行われる。発行された場合、制御はステップ７１２（図１４）に移行し、ライト動作が、既に割り当てられているデフォルト・バッファの１つに影響を与えるか否かについて判定を行う。バス・マスタ１０８が、デフォルト・バッファ２１０の内の割り当てらた１つであり、ライト動作が、コンフィギュレーション・ライトまたはコマンドのように、バス・マスタ１０８に対して実行される場合、新しいプロセスが開始された可能性があり、この場合、デフォルト・バッファ２１０内にある先取りされたデータはいずれもが古い又は一致しない恐れがある。かかる場合、バッファはライト・コマンドによる影響を受けたと言う。このような場合、制御はステップ７１４に移行し、ライト動作によって影響を受けたスロット又はマスタに対応するデフォルト・バッファ２１０をフラッシュする。次に、制御はステップ７１４からステップ７１６に移行する。ステップ７１２において、ライト・コマンドがいずれのデフォルト・バッファ２１０にも影響を与えないと判定された場合、制御はステップ７１２からステップ７１６に移行する。ステップ７１６において、あらゆるライト・データのライト・ポスティング・バッファへの受け取り及びBUFF_VALID_PNTRの更新（ステップ７１８）を含む、残りのライト動作処理が行われる。BUFF_VALID_PNTRは、デフォルト・アルゴリズムおよび保存アルゴリズムについてのみ更新される。これで、このルーチンは終了し、それをコールしたいずれかのプロセスに戻る。

【００６２】ステップ７０２に戻り、動作がライト動作でないと判定された場合、制御はステップ７０４に移行し、動作が予約コマンドであるか否かについて判定が行われる。予約コマンドである場合、制御はステップ７２０（図１５）に移行し、予約コマンドがブリッジ１０６によって受け取られる。次に、制御はステップ７２２に移行し、予約資源２２０の１つがこの要求のために使用可能であるか否かについて判定が行われる。使用可能ではない場合、制御はステップ７４２に移行し、要求元バス・マスタにリトライを発行し、このルーチンは終了する。予約資源２２０が使用可能である場合、制御はステップ７２２からステップ７２４に移行する代わりに、予約コマンド内に含まれているデータ（即ち、アドレスおよびカウント）を用いて、予約資源レジスタを初期設定する。次に、制御はステップ７２８に移行し、３つの１２８バイト・バッファをこの要求に割り当て、ステップ７３０に移行し、これらのバッファに対するステータスをEMPTYに変更する。次に、制御はステップ７３０からステップ７３２に移行し、ブリッジ１０６はデータの３ないし４ラインを、バッファ２０６の第１の１２８バイト・バッファに先取りする。次に、制御はステップ７３４に移行し、最後のバッファ・ポインタ(LAST_BUFF_PNTR)を初期設定し、どのバッファが次に書き込まれるのかを示す。

【００６３】次に、制御はステップ７３６に移行し、送出コンフィギュレーション・ビット(forwarding configuration bit)がセットされているか否かについて判定が行われる。セットされている場合、次に制御はステップ７３８に移行し、予約コマンドをいずれかの上流ブリッジに向けて送出する。次に、制御はステップ７３８からステップ７４０に移行する。ステップ７３６において、コンフィギュレーション・ビットがセットされていない場合、制御は直接ステップ７４０に移行する。ステップ７４０において、予約タイマをリセットし、開始する。２５ｍｓ、５０ｍｓまたは１００ｍｓのような所定の時間量の後、予約資源にアクセスされなかった場合、予約タイマはこの予約を取り消す。この無動作期間の後、データは古くなったものと見なす。これで、このルーチンは終了する。

【００６４】ステップ７０４に戻り、動作が予約コマンドではないと判定された場合、次に制御はステップ７０６に移行し、動作はバス・マスタからのリード・リトライか否かについて判定が行われる。バス・マスタからのリード・リトライでない場合、この動作は最初のリード動作であると見なされ、制御はステップ７４４（図１７）に移行する。

【００６５】ステップ７４４において、要求元バス・マスタが、それにデフォルト・バッファ２１０を既に割り当ててあるか、または新しいバス・マスタであるかについて判定が行われる。これが新しいマスタでないと判定された場合、制御はステップ７４６に移行し、要求されたアドレスが、BUFF_OUT_PNTRによって指し示されているアドレスに等しいか否かについて判定が行われる。等しい場合、制御は次にステップ７５０に移行する。等しくない場合、制御は次にステップ７４８に移行し、マスタは既に、新しい、連続しないアドレスからリードを開始しているので、割り当てられたデフォルト・バッファ２１０をフラッシュする。次に、制御はステップ７５０に移行し、デフォルト・バッファ２１０をこのマスタに再度割り当て、上述のように、バッファ・レジスタを初期設定する。次いで、制御はステップ７６８に移行する。

【００６６】ステップ７４４において、これが新しいマスタであると判定された場合、制御はステップ７５６に移行し、デフォルト・バッファ２１０のいずれかが現在EMPTY（空）と示されているか否かについて判定が行われる。EMPTYと示されているデフォルト・バッファがあ

る場合、制御はステップ７５８に移行し、EMPTYのデフォルト・バッファ２１０の１つを新しいマスタに割り当てる。ステップ７５８から、制御はステップ７６８に移行する。EMPTYと示されているデフォルト・バッファがない場合、制御はステップ７６０に移行し、いずれかのデフォルト・バッファ２１０に、現在REQUESTまたはPART_COMPと示されているものがあるか否かについて判定が行われる。かかるバッファがある場合、制御はステップ７２６に移行し、PREFETCH_FLAGがセットされているか否かについて判定が行われ、セットされている場合、制御はステップ７６４に移行し、デフォルト・バッファ２１０が新しいマスタに割り当てられる。次に、制御はステップ７６４からステップ７６８に移行する。ステップ７６０および７６２において、判定が否定的である場合、制御はステップ７６６に移行し、バス・マスタに、後の時点で動作をリトライするように要求する。

【００６７】ステップ７６８において、制御は、新しく割り当てられたまたは再割り当てされたデフォルト・バッファ２１０に対する先取りをディスエーブルする。次に、制御はステップ７７０（図１６）に移行し、リード要求情報を格納する。次に、制御はステップ７７２に移行し、ブリッジ１０６が宛先バス上のターゲットから要求されたデータを集めている間に、バス・マスタに動作をリトライするように要求する。次に、制御はステップ７７４に移行し、リード要求がメモリ・リード多重コマンド（memory read multiple command）であったか否かについて判定を行う。そうであった場合、制御はステップ７７６に移行し、ブリッジ１０６は、主メモリ１０４からデータの３ないし４ラインを読み取る。次に、制御はステップ７７６からステップ７８４に移行する。ステップ７７４において、リード・コマンドがメモリ・リード多重でなかったと判定された場合、制御はステップ７７８に移行し、リード・コマンドはメモリ・リード・ライン・コマンドであったか否かについて判定を行う。そうであった場合、制御はステップ７８０に移行し、ブリッジ１０６が主メモリ１０４からデータの１行を読み取り、次いで、制御はステップ７８４に移行する。ステップ７７８において、コマンドがメモリ・リード・ライン・コマンドでなかったと判定された場合、制御はステップ７８２に移行し、ブリッジ１０６は主メモリ１０４からデータの１Dwordを読み取り、次いで、制御はステップ７８４に移行する。

【００６８】ステップ７８４において、ブリッジ１０６は主メモリ１０４からデータを受け取り、ステップ７８６において、ブリッジ１０６は受け取ったデータを割り当てられたバッファに格納し、処理は終了する。このルーチンは後に要求元バス・マスタがこの動作のリトライを行うときにコールされる。

【００６９】ここでステップ７０６に戻り、動作がバス・マスタからのリード・リトライ動作であると判定された場合、制御はステップ７０８に移行し、更に、このバス・マスタは以前にバッファ予約を要求したことがあるか否かについて判定が行われる。要求したことがある場合、制御はステップ７８８（図１８）に移行し、このリトライ・リード動作に対するアドレスが、予約資源２２０の制限内であるか否かについて判定が行われる。制限内でない場合、制御はステップ７９０に移行して予約を取り消し、制御はステップ７４４（図１７）に移行し、デフォルト・アルゴリズムを用いてリード動作を開始する。ステップ７８８において、リード動作が予約制限内であることが判定された場合、制御はステップ７９２に移行し、次に、リード・アドレスが３つのバッファの制限の１つの中にあるか否かについて判定が行われる。そうである場合、制御はステップ７９４に移行し、要求されたデータのDwordをバス・マスタに供給する。ステップ７９４から、制御はステップ７９６に移行し、バス・マスタがこの特定の１２８バイト・バッファからリードを行ったのは、これが最初であるか否かについて判定が行われる。

【００７０】これが最初である場合、制御はステップ７９８に移行し、ブリッジ１０６が、LAST_BUFF_PNTRによって示される次のバッファにデータを先取りする。バッファ２０６は、リースト・リーセントリー・ユーズド（least recently used）（ＬＲＵ）アルゴリズムにしたがって充填される。一旦あるバッファからデータが最初に読み出されたなら、この次のデータ先取り動作には、最も長い間使用されていないバッファが用いられる。したがって、この点において、バス・マスタは１つのバッファからリードを行い、次のバッファは先取りされたデータを保持し、別のバッファが以前に先取りされたデータを保持する。したがって、バス・マスタがバックアップする必要があっても、あるいはその現アドレスからステップ・バックする必要があっても、データは、以前に先取りされたバッファにある可能性が未だに高い。制御はステップ７９８からステップ８００に移行する。これが最初ではない場合、制御はステップ７９６からステップ８００に移行し、バス・マスタによるリード動作がバースト・リードであるか否かについて判定が行われる。バースト・リードである場合、制御はステップ７８８に戻り、追加データを供給する。バースト・リードでない場合、制御は終了に移行する。

【００７１】ステップ７９２において、リード動作がバッファ制限内にないことが判定された場合、制御はステップ８０４に移行し、ブリッジ１０６は、ブリッジが要求されたデータを取り出している間に、マスタに動作をリトライすることを強制する。次に、制御はステップ８０４から８０６に移行し、ブリッジが主メモリ１０４からデータを読み取り、ステップ８０８に移行してそのデータを受け取り、ステップ８１０に移行して受け取ったデータをバッファ２０６に格納し、ブリッジはバス・マ

スタがリード動作をリトライするのを待つ。

【００７２】ステップ７０８に戻り、このリード動作のために以前に要求した予約資源がなかった場合、制御はステップ８１２のデフォルト・アルゴリズム（図１９）に移行する。

【００７３】ステップ８１２において、デフォルト・バッファ２１０の１つから、要求されたデータをバス・マスタに供給する。次いで、制御はステップ８１４に移行し、リード動作がバースト動作であるか否かについて判定が行われる。バースト動作でない場合、処理はステップ８１５に進み、保存（conservative）アルゴリズムが選択されるか否かについて判定が行われる。される場合、制御はステップ８１７に移行し、バッファをフラッシュする。保存アルゴリズムが選択されない場合、制御は終了に移行する。ステップ８１４において、バースト動作が進行中である場合、制御はステップ８１６に移行し、次の要求されたアドレスが、データ・バッファ内に含まれているか否かについて判定が行われる。そうである場合、制御はステップ８１８に移行し、データをバス・マスタに供給する。次に、制御はステップ８１４に戻り、要求されれば、追加データを供給する。ステップ８１６において、次の要求されたアドレスがバッファ内に含まれていないと判定された場合、制御はステップ８２０に移行し、このバッファに対して先取りがイネーブルされているか否かについて判定が行われる。イネーブルされていない場合、制御はステップ８３２に移行し、先取りをイネーブルする。次いで、上述のように、制御はステップ７７０に移行する。

【００７４】ステップ８２０において、先取りがイネーブルされていると判定された場合、制御はステップ８２２に移行し、リード・トランザクション情報をブリッジ１０６に格納する。次に、制御はステップ８２４および８２６に移行し、ブリッジ１０６がバス・マスタにリトライを発行し、ブリッジはリード動作を主メモリ１０４に発行する。この点以降、このバス・マスタに対して先取りがイネーブルされる。ステップ８２８において、ブリッジ１０６はリード・データを受け取り、ステップ８３０において、ブリッジは受け取ったデータをバッファに格納し、バス・マスタがリード動作をリトライするのを待つ。

【００７５】以上の本発明の開示および記載は例示であり、その説明に供するものであり、本発明の精神から逸脱することなく、サイズ、形状、材質、構成要素、回路素子、配線接続および接点、ならびに図示した回路や構成、更に動作方法における種々の変更が可能である。

【図面の簡単な説明】

【図１】好適実施例に従ったブリッジ・デバイスを組み込んだコンピュータ・システムのブロック図である。

【図２】好適実施例に従ったブリッジ・デバイスを組み込んだコンピュータ・システムの更に詳細なブロック図である。

【図３】好適実施例に従ったブリッジ・デバイスの機能ブロックを示すブロック図である。

【図４】好適実施例に従ったブリッジ・デバイスの機能ブロックを示すブロック図である。

【図５】好適実施例に従った２つの予約ブリッジ間に結合された非予約ブリッジを有する、バスの階層の一例を示すブロック図である。

【図６】非予約ブリッジが階層の最上部のホスト・ブリッジとして動作する、バスの階層の一例を示すブロック図である。

【図７】予約コマンドを組み込んだ、ＰＣＩバス・リード・トランザクションを示すタイミング図である。

【図８】予約コマンドを組み込んだ６４ビット・アドレスによる、ＰＣＩバス・リード・トランザクションを示すタイミング図である。

【図９】バッファのブロック図であり、Ａは好適実施例に従った単一の１２８バイト・データ・バッファのブロック図であり、Ｂは好適実施例に従った予約バッファのブロック図である。

【図１０】好適実施例に従ったバッファ状態を示す状態図である。

【図１１】好適実施例に従ったバッファ先取り状態を示す状態図である。

【図１２】好適実施例に従ったプリフェッチ及びフラッシュ・アルゴリズムを示すフローチャートである。

【図１３】好適実施例に従った、図１２の「Ａ」に続くフローチャートである。

【図１４】好適実施例に従った、図１２の「Ｂ」に続くフローチャートである。

【図１５】好適実施例に従った、図１２の「Ｃ」に続くフローチャートである。

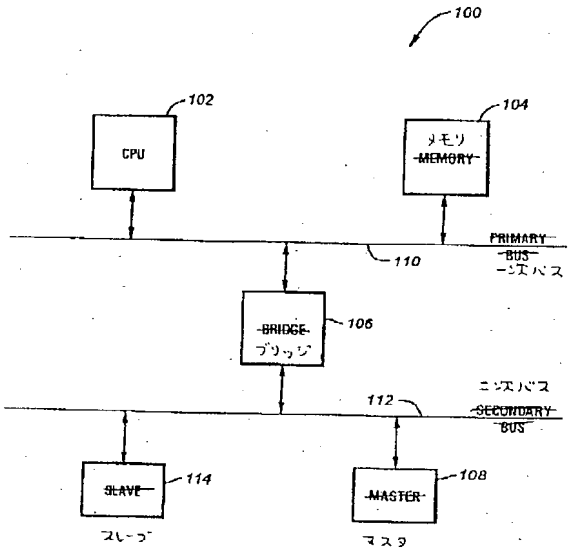
【図１６】好適実施例に従った、図１７および図１９の「Ｇ」に続くフローチャートである。

【図１７】好適実施例に従った、図１２の「Ｄ」に続くフローチャートである。

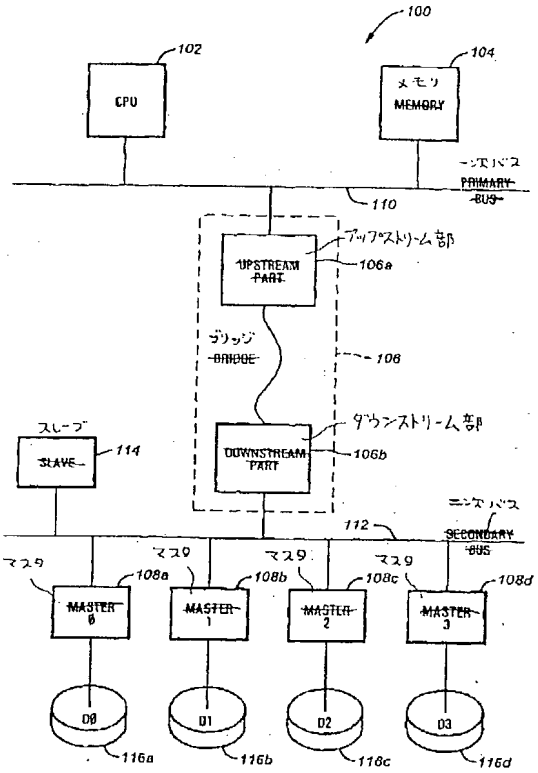
【図１８】好適実施例に従った、図１２の「Ｅ」に続くフローチャートである。

【図１９】好適実施例に従った、図１２の「Ｆ」に続くフローチャートである。

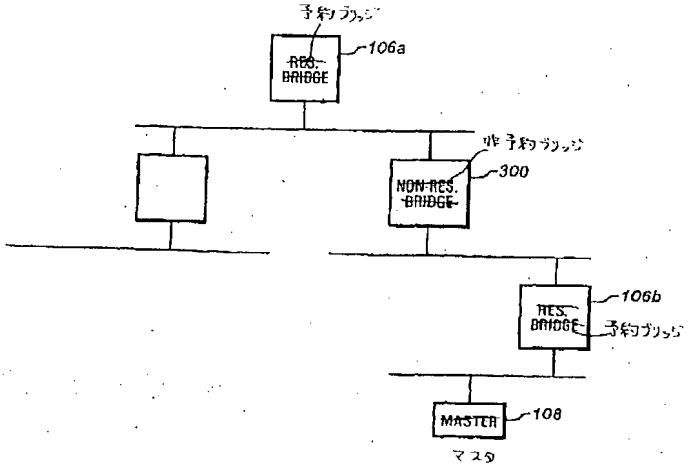
【図1】



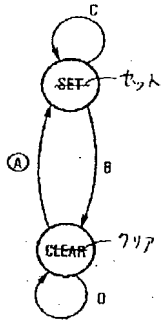
【図2】



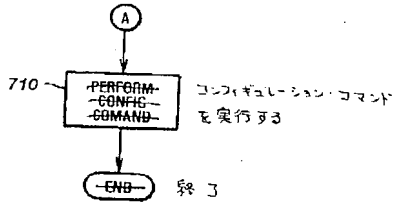
【図5】



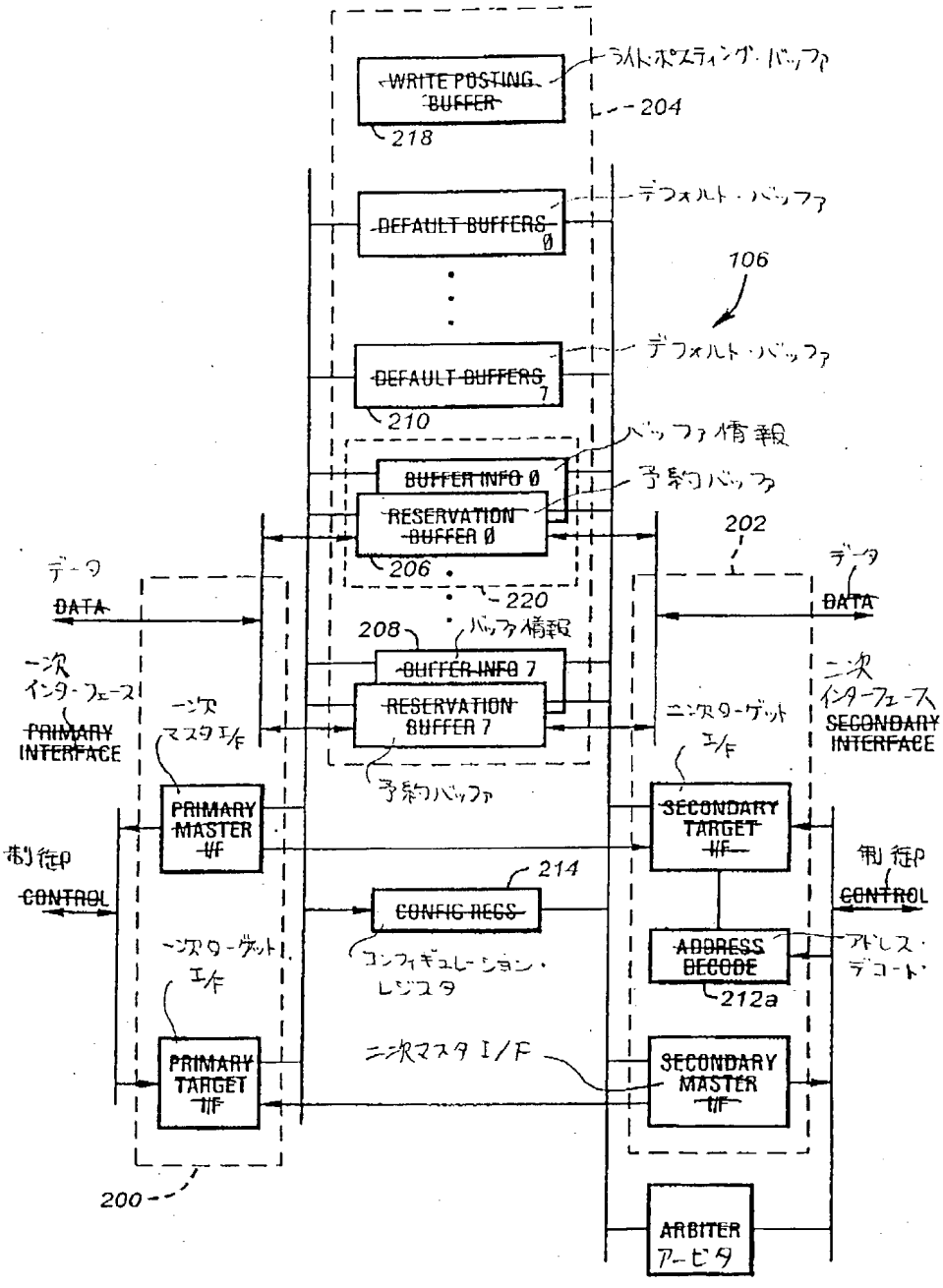
【図11】



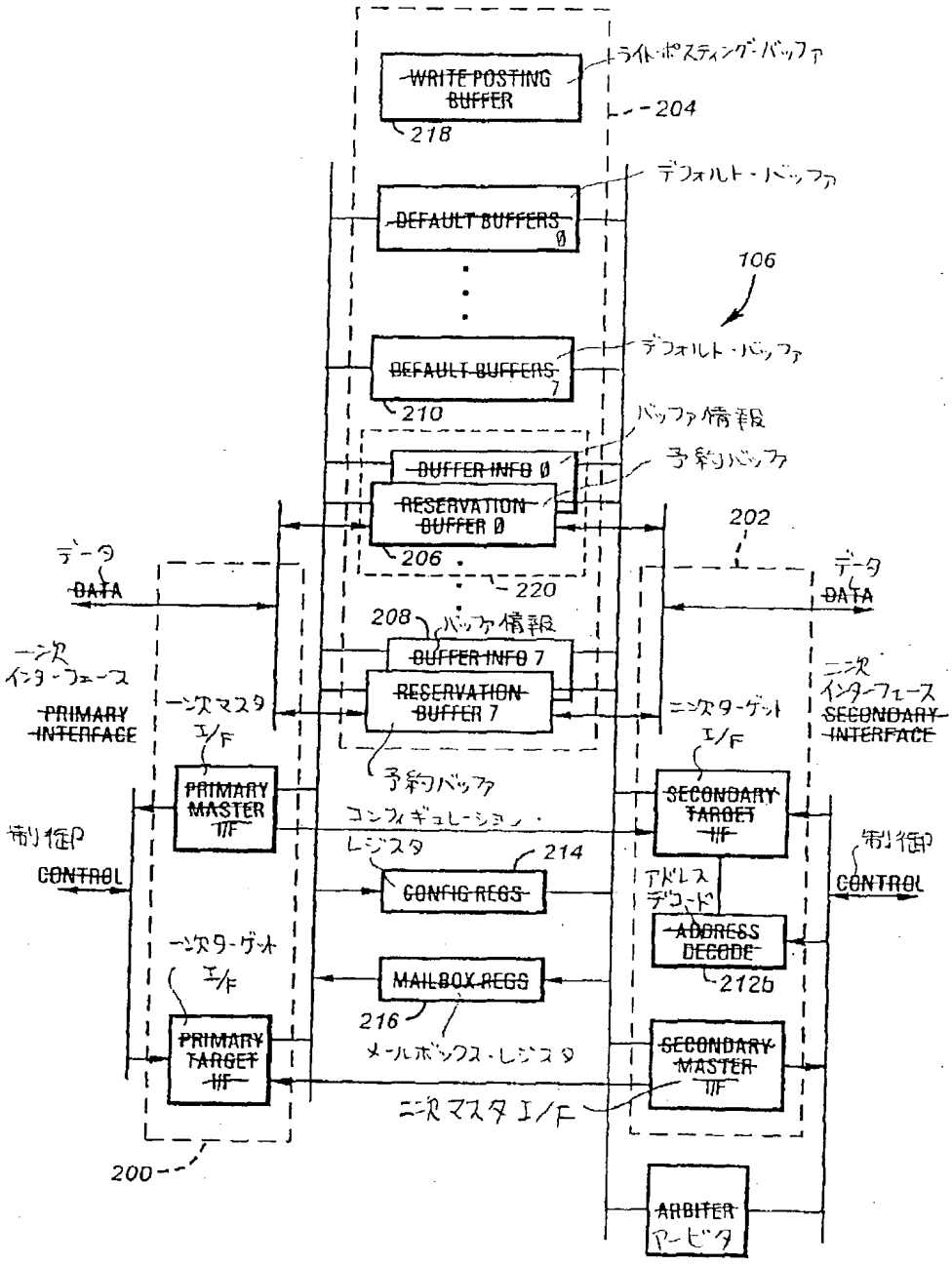
【図13】



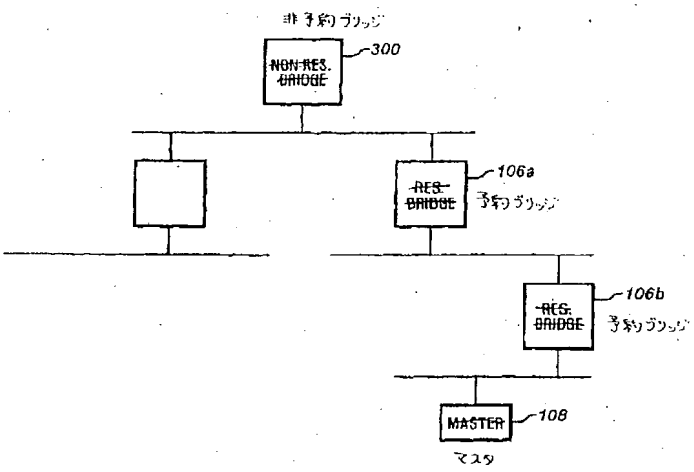
【図3】


WRITE POSTING BUFFER
ライトポスティング・バッファ
204
218
DEFAULT BUFFERS 0
デフォルト・バッファ
106
DEFAULT BUFFERS 7
デフォルト・バッファ
210
バッファ情報
BUFFER INFO 0
予約バッファ
RESERVATION BUFFER 0
206
220
202
データ
DATA
データ
DATA
208
バッファ情報
BUFFER INFO 7
一次インターフェース
PRIMARY INTERFACE
一次マスタI/F
RESERVATION BUFFER 7
二次ターゲットI/F
二次インターフェース
SECONDARY INTERFACE
予約バッファ
PRIMARY MASTER I/F
SECONDARY TARGET I/F
制御
CONTROL
214
CONFIG REGS
制御
CONTROL
一次ターゲットI/F
コンフィギュレーション・レジスタ
ADDRESS DECODE
アドレス・デコード
212a
PRIMARY TARGET I/F
二次マスタI/F
SECONDARY MASTER I/F
200
ARBITER
アービタ

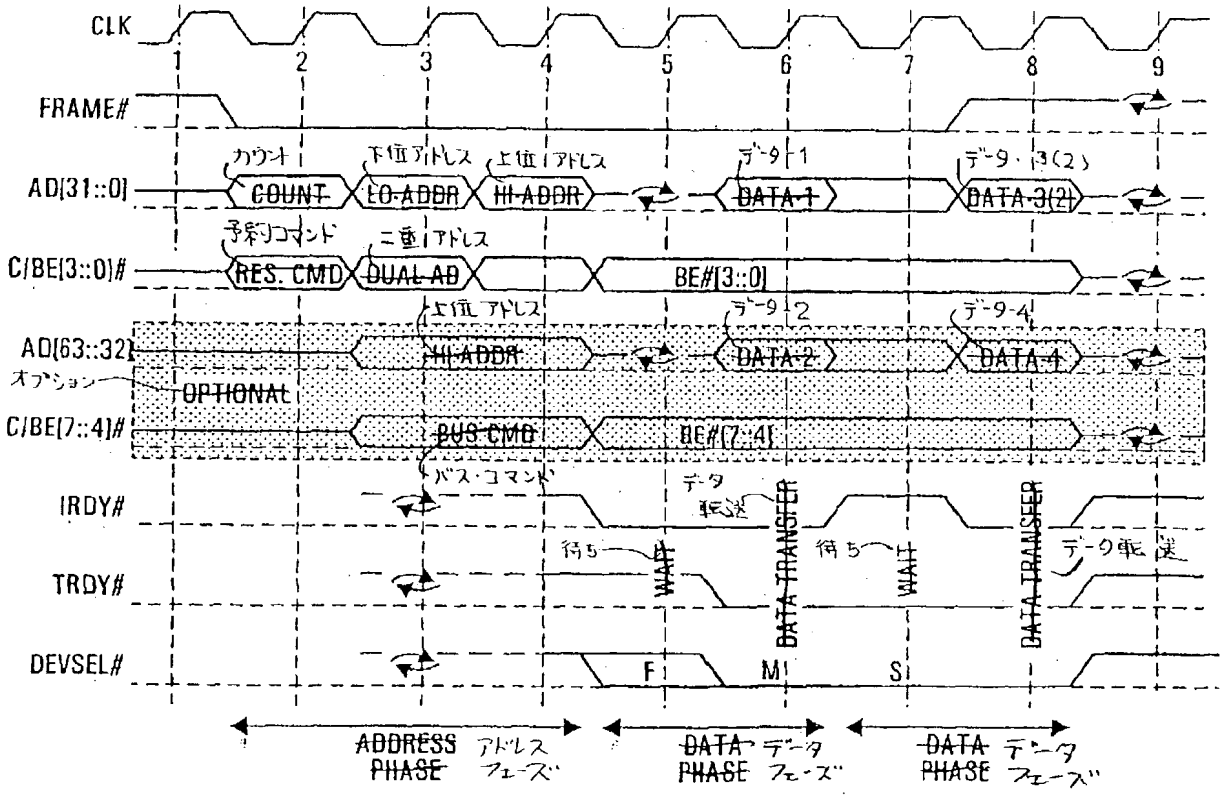
【図4】


WRITE POSTING BUFFER
ライト・ポスティング・バッファ
204
218
DEFAULT BUFFERS 0
デフォルト・バッファ
106
DEFAULT BUFFERS 7
デフォルト・バッファ
210
バッファ情報
予約バッファ
BUFFER INFO 0
RESERVATION BUFFER 0
206
202
220
データ
DATA
208
BUFFER INFO 7
RESERVATION BUFFER 7
一次インターフェース
PRIMARY INTERFACE
一次マスタ I/F
一次ターゲット I/F
二次インターフェース
SECONDARY INTERFACE
PRIMARY MASTER I/F
SECONDARY TARGET I/F
コンフィギュレーション・レジスタ
214
CONFIG REGS
制御
CONTROL
アドレスデコード
ADDRESS DECODE
212b
MAILBOX REGS
216
メールボックス・レジスタ
PRIMARY TARGET I/F
SECONDARY MASTER I/F
二次マスタ I/F
200
ARBITER
アービタ

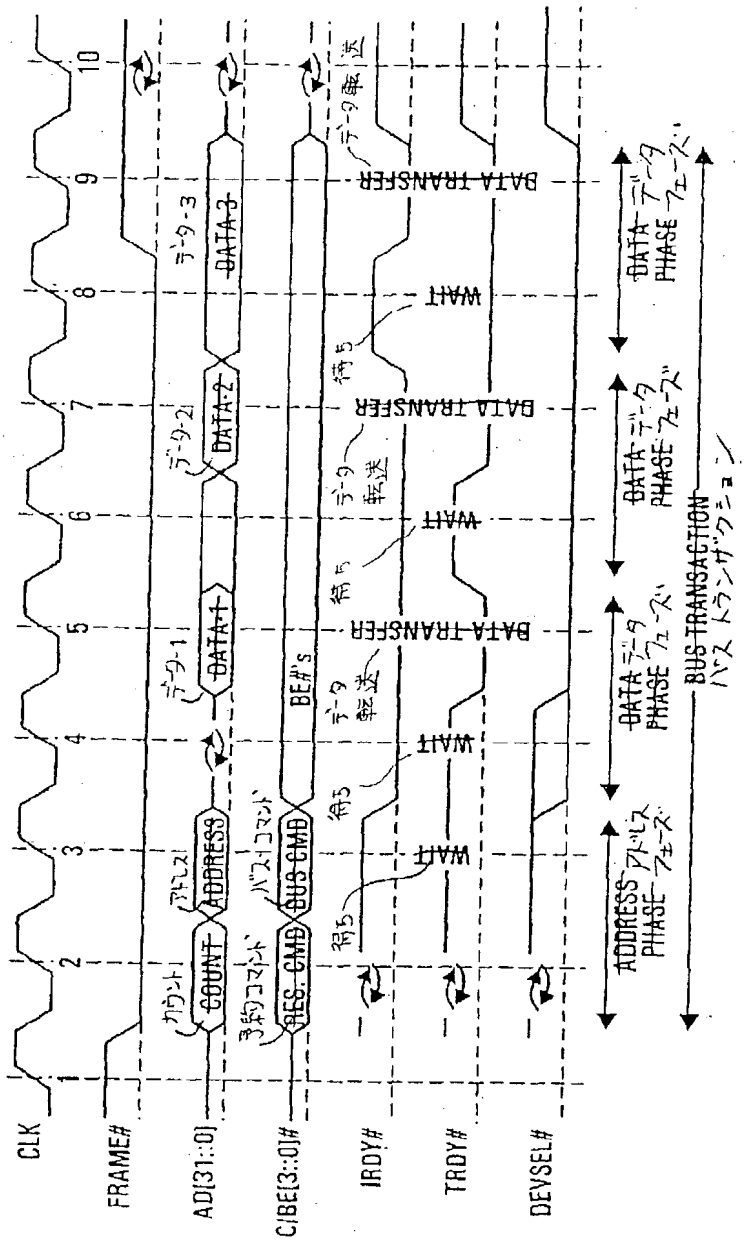
【図6】


非予約ブリッジ
NON-RES. BRIDGE
300
RES. BRIDGE
106a
予約ブリッジ
RES. BRIDGE
106b
予約ブリッジ
MASTER
108
マスタ

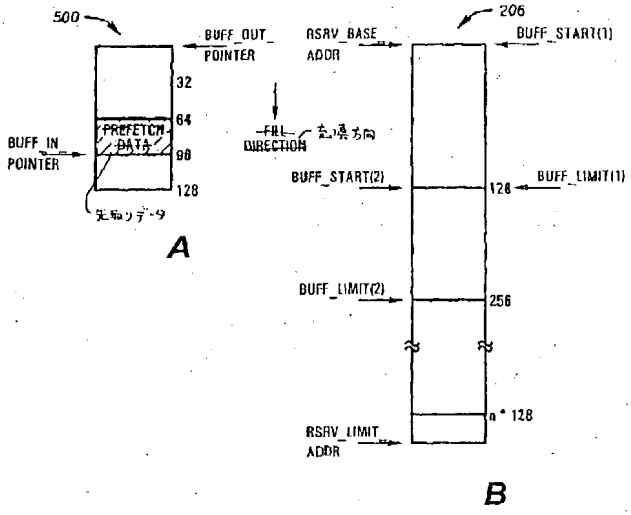
【図7】


CLK
1
2
3
4
5
6
7
8
9
FRAME#
AD[31::0]
カウント
COUNT
下位アドレス
LO-ADDR
上位アドレス
HI-ADDR
データ-1
DATA-1
データ-3(2)
DATA-3(2)
C/BE[3::0]#
予約コマンド
RES. CMD
二重アドレス
DUAL AD
BE#[3::0]
AD[63::32]
上位アドレス
HI-ADDR
データ-2
DATA-2
データ-4
DATA-4
オプション
OPTIONAL
C/BE[7::4]#
BUS CMD
BE#[7::4]
バス・コマンド
IRDY#
データ転送
TRDY#
待ち
WAIT
DATA TRANSFER
待ち
WAIT
DATA TRANSFER
データ転送
DEVSEL#
F
M
S
ADDRESS PHASE
アドレスフェーズ
DATA PHASE
データフェーズ
DATA PHASE
データフェーズ

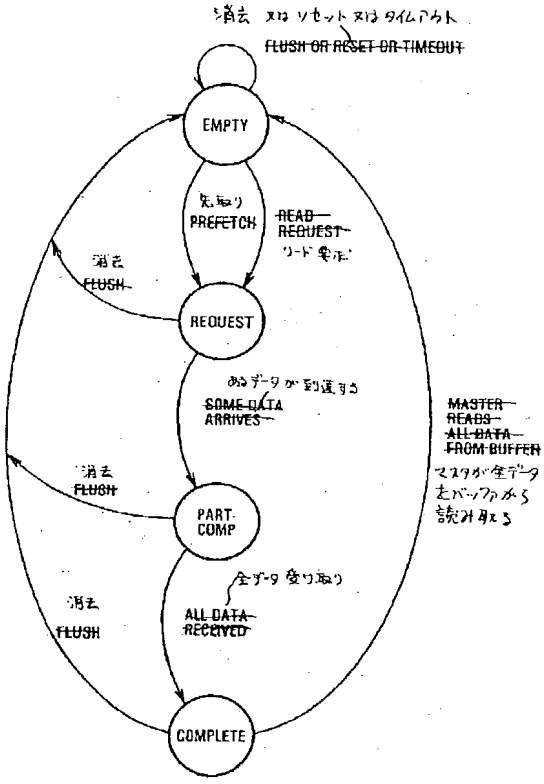
【図8】


CLK
FRAME#
AD[31::0]
C/BE[3::0]#
IRDY#
TRDY#
DEVSEL#
1
2
3
4
5
6
7
8
9
10
COUNT
ADDRESS
RES. CMD
BUS CMD
BE#'s
DATA-1
DATA-2
DATA-3
DATA TRANSFER
WAIT
ADDRESS PHASE
DATA PHASE
BUS TRANSACTION

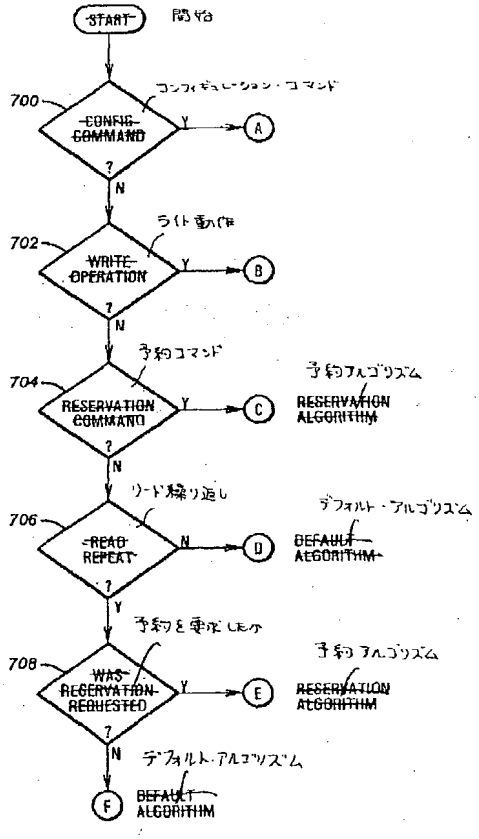
【図9】



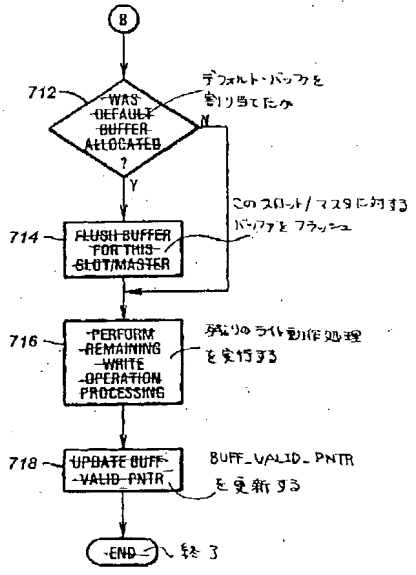
【図10】



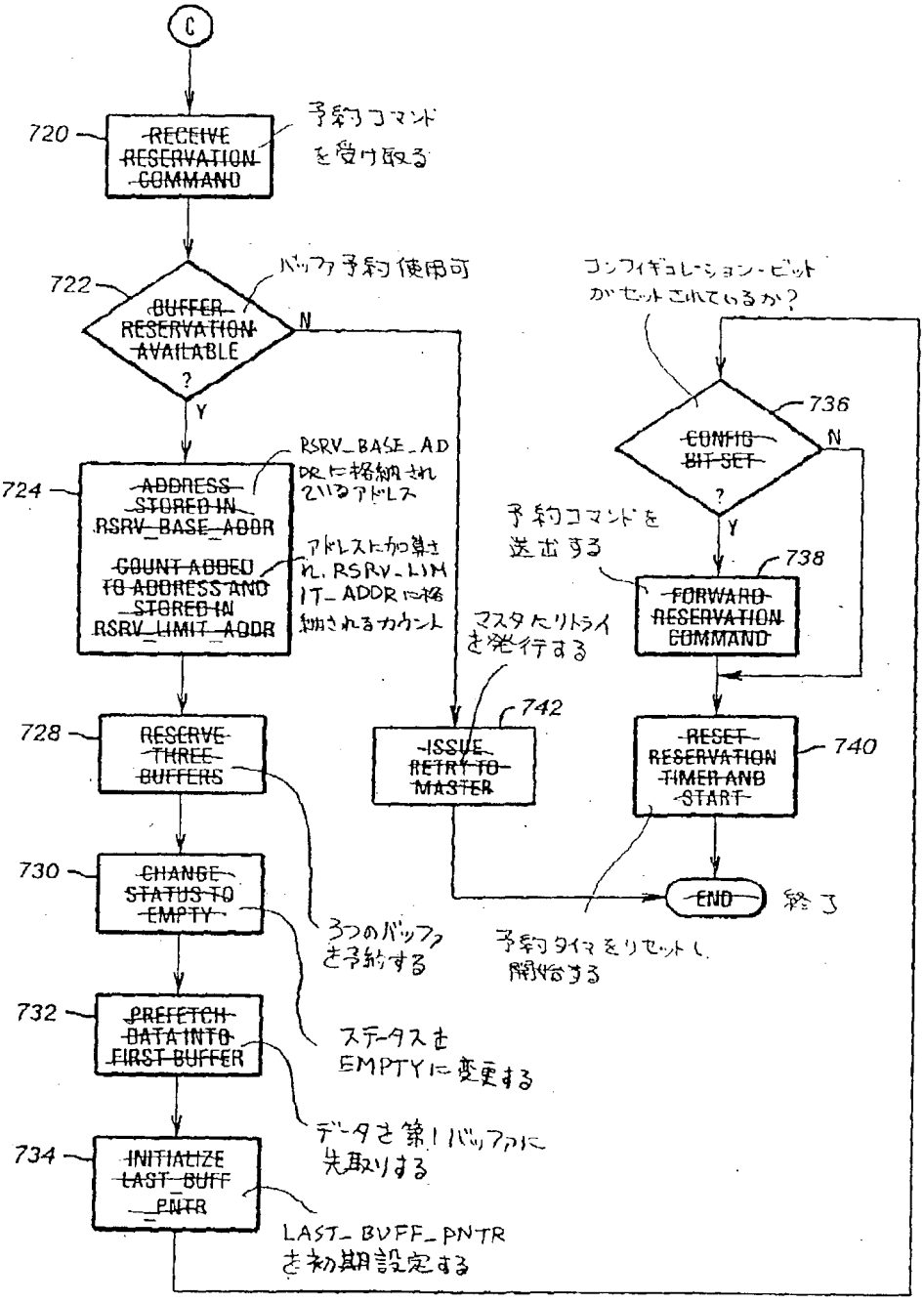
【図12】



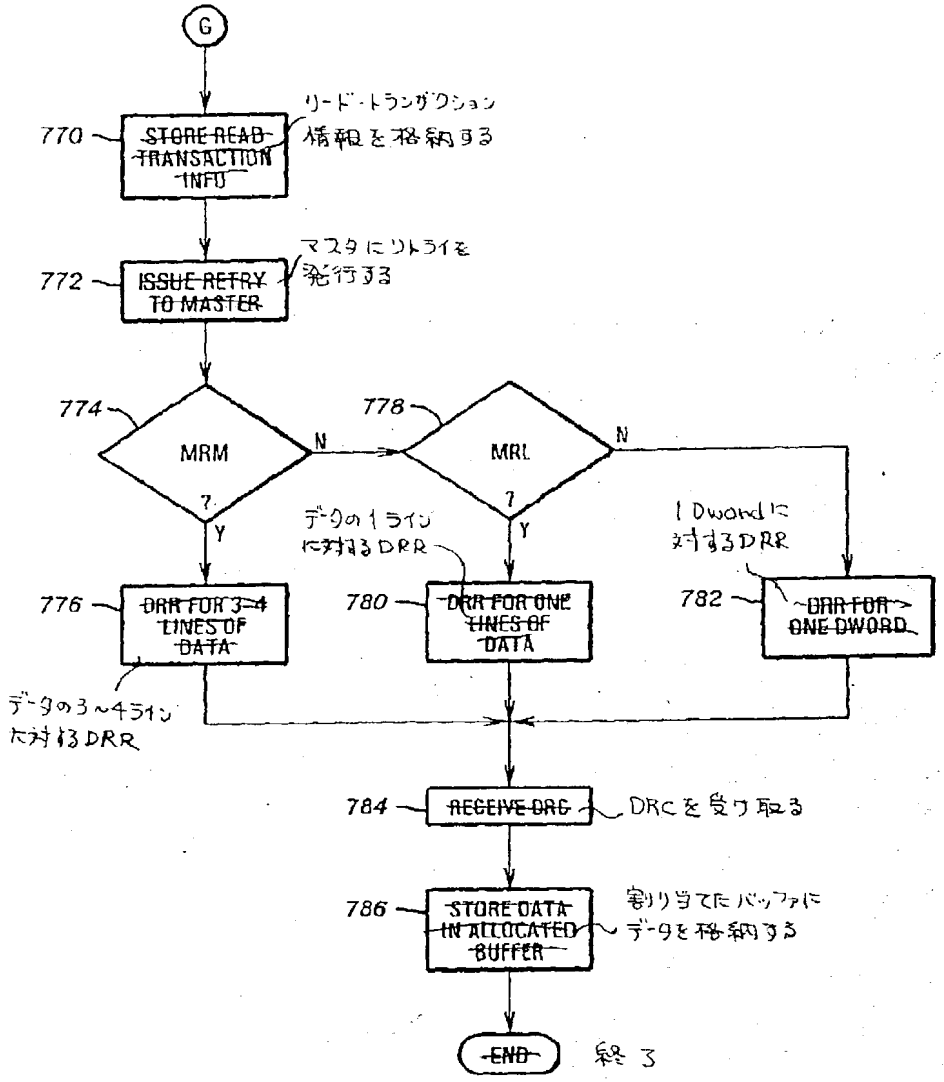
【図14】



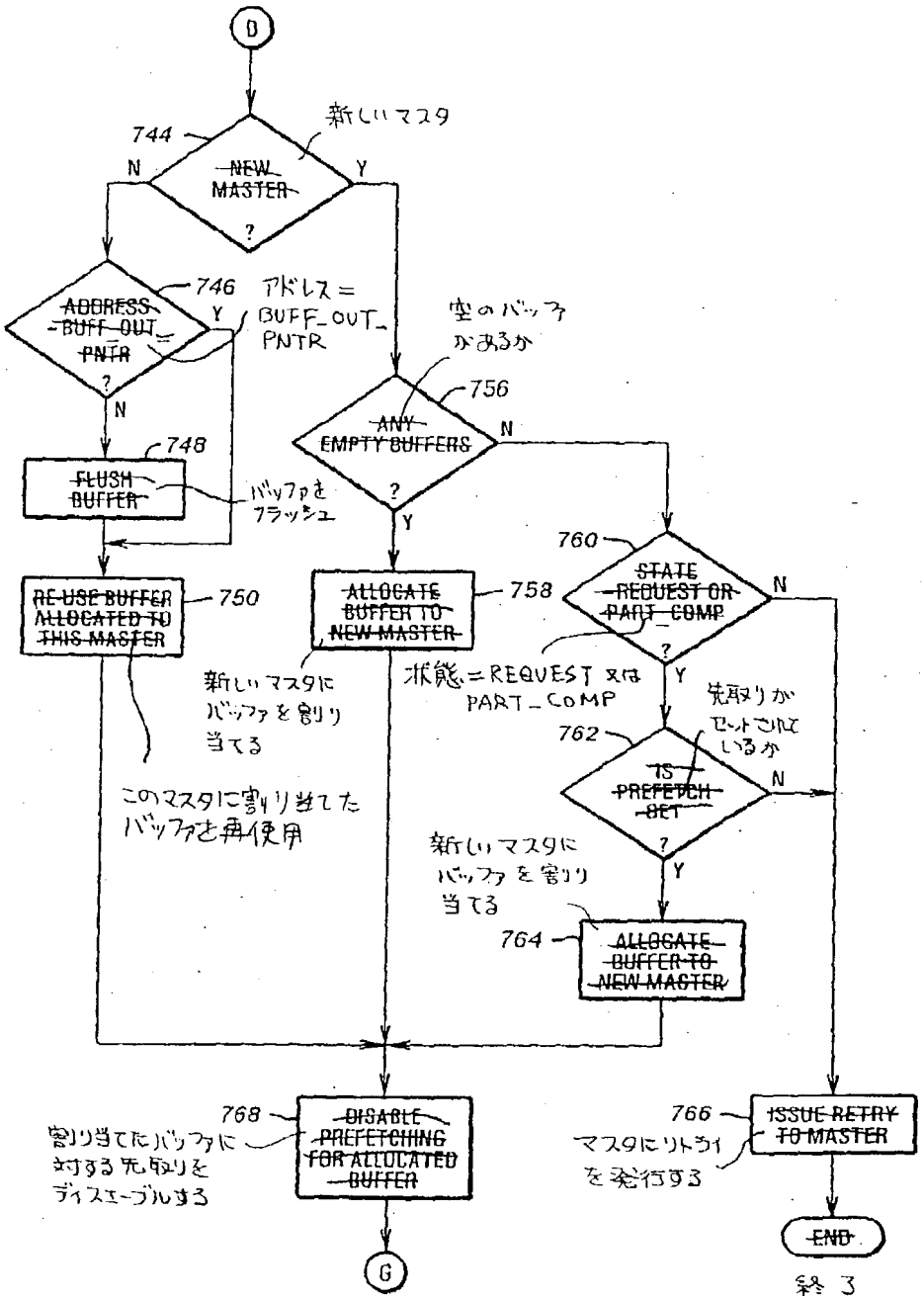
【図15】


C
720
RECEIVE RESERVATION COMMAND
予約コマンドを受け取る
722
BUFFER RESERVATION AVAILABLE ?
バッファ予約使用可
N
Y
724
ADDRESS STORED IN RSRV_BASE_ADDR
RSRV_BASE_ADDRに格納されているアドレス
COUNT ADDED TO ADDRESS AND STORED IN RSRV_LIMIT_ADDR
アドレスに加算され、RSRV_LIMIT_ADDRに格納されるカウント
728
RESERVE THREE BUFFERS
3つのバッファを予約する
730
CHANGE STATUS TO EMPTY
ステータスをEMPTYに変更する
732
PREFETCH DATA INTO FIRST BUFFER
データを第1バッファに先取りする
734
INITIALIZE LAST_BUFF_PNTR
LAST_BUFF_PNTRを初期設定する
コンフィギュレーション・ビットがセットされているか？
736
CONFIG BIT SET ?
N
Y
予約コマンドを送出する
738
FORWARD RESERVATION COMMAND
740
RESET RESERVATION TIMER AND START
予約タイマをリセットし開始する
742
ISSUE RETRY TO MASTER
マスタにリトライを発行する
END
終了

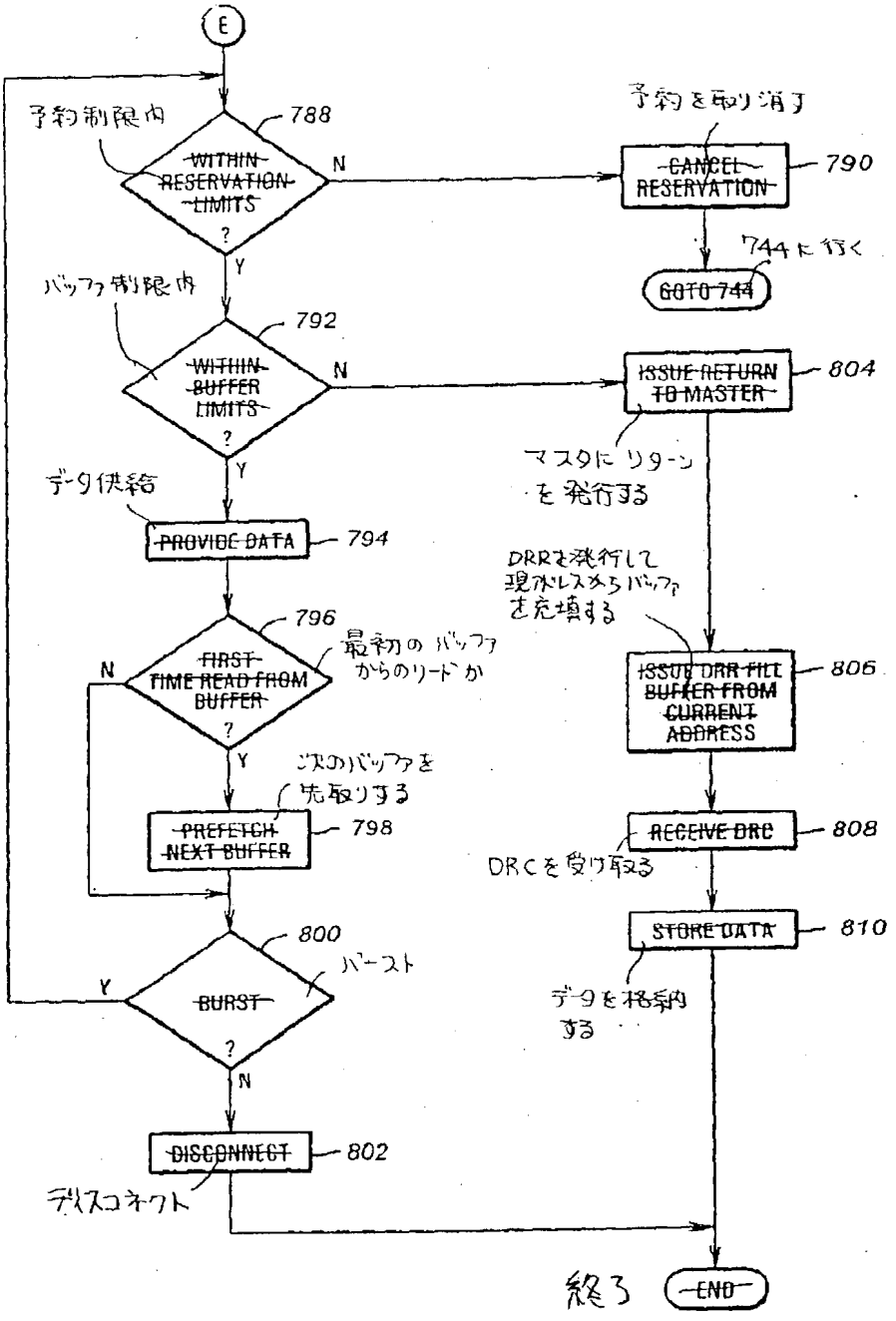
【図１６】


G
770
STORE READ TRANSACTION INFO
リード・トランザクション情報を格納する
772
ISSUE RETRY TO MASTER
マスタにリトライを発行する
774
MRM
?
N
Y
778
MRL
?
N
Y
776
DRR FOR 3-4 LINES OF DATA
データの3～4ラインに対するDRR
780
DRR FOR ONE LINES OF DATA
データの1ラインに対するDRR
782
DRR FOR ONE DWORD
1 Dwordに対するDRR
784
RECEIVE DRC
DRCを受け取る
786
STORE DATA IN ALLOCATED BUFFER
割り当てたバッファにデータを格納する
END
終了

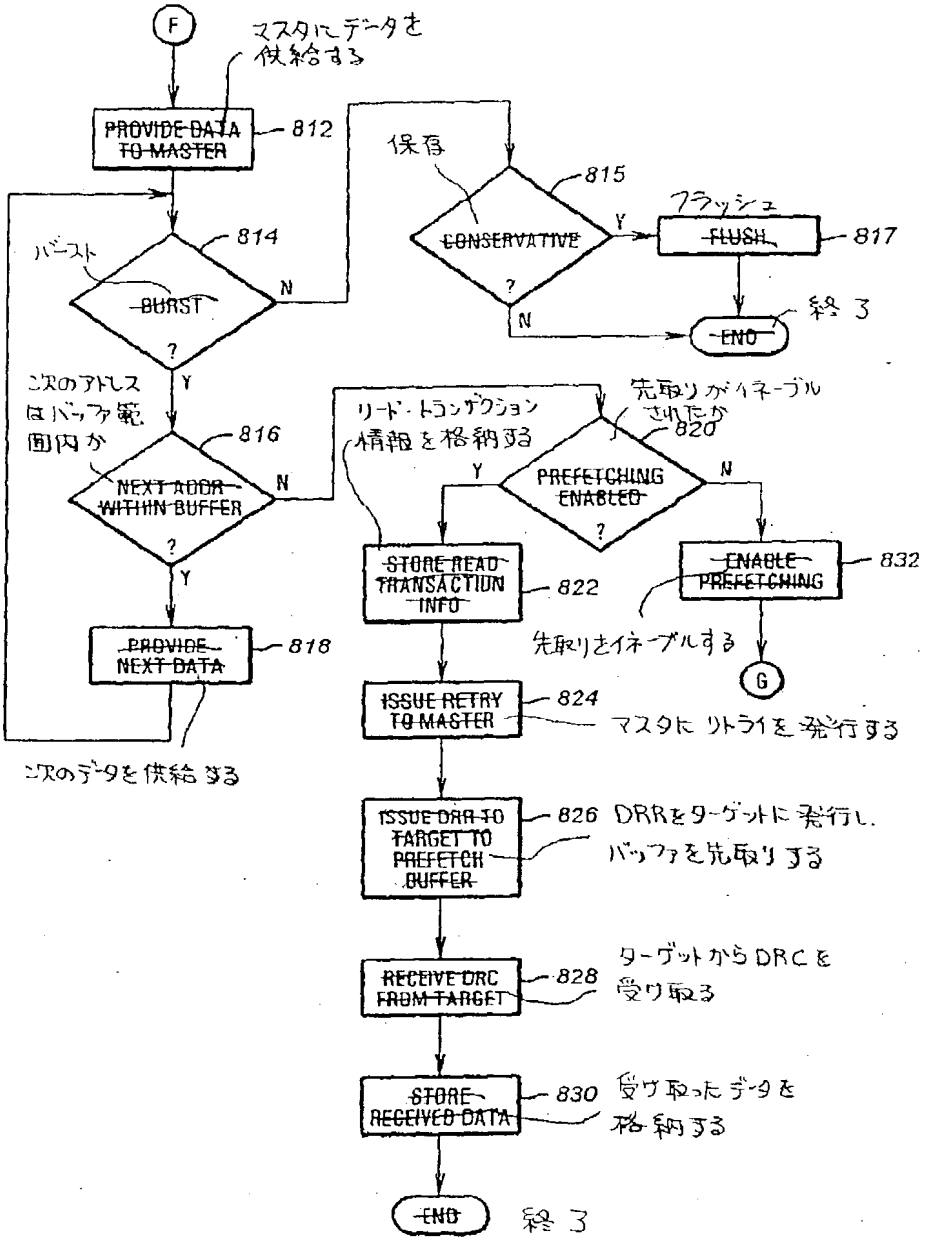
【図17】


D
744
新しいマスタ
NEW MASTER
N
Y
?
746
アドレス＝
BUFF_OUT_PNTR
ADDRESS BUFF_OUT_PNTR
Y
?
N
748
FLUSH BUFFER
バッファをフラッシュ
空のバッファがあるか
756
ANY EMPTY BUFFERS
N
?
Y
760
STATE REQUEST OR PART_COMP
N
?
Y
状態＝REQUEST又はPART_COMP
RE-USE BUFFER ALLOCATED TO THIS MASTER
750
このマスタに割り当てたバッファを再使用
ALLOCATE BUFFER TO NEW MASTER
758
新しいマスタにバッファを割り当てる
762
IS PREFETCH SET
先取りがセットされているか
N
?
Y
新しいマスタにバッファを割り当てる
764
ALLOCATE BUFFER TO NEW MASTER
768
DISABLE PREFETCHING FOR ALLOCATED BUFFER
割り当てたバッファに対する先取りをディスエーブルする
766
ISSUE RETRY TO MASTER
マスタにリトライを発行する
END
終了
G

【図18】


E
予約制限内
788
WITHIN RESERVATION LIMITS
?
N
Y
予約を取り消す
CANCEL RESERVATION
790
744に行く
GOTO 744
バッファ制限内
792
WITHIN BUFFER LIMITS
?
N
Y
ISSUE RETURN TO MASTER
804
マスタにリターンを発行する
データ供給
PROVIDE DATA
794
DRRを発行して現アドレスからバッファを充填する
796
FIRST TIME READ FROM BUFFER
?
N
Y
最初のバッファからのリードか
次のバッファを先取りする
PREFETCH NEXT BUFFER
798
ISSUE DRR FILL BUFFER FROM CURRENT ADDRESS
806
RECEIVE DRC
808
DRCを受け取る
STORE DATA
810
データを格納する
800
BURST
?
バースト
Y
N
DISCONNECT
802
ディスコネクト
終了
END

【図19】



フロントページの続き

(71)出願人　591030868
20555 State Highway 249, Houston, Texas 77070, United States of America

【外国語明細書】

1. Title of Invention
Buffer Reservation Method for a Bus Bridge System

2. Claims

1. A method of communication between a bus bridge and a bus master, the bus bridge having at least one data buffer, the method comprising the steps of:

(a) receiving a reservation command from the bus master; and

(b) reserving a data buffer for the bus master in response to said reservation command.

2. The method of claim 1, further comprising the step of:

(c) receiving a bus command after step (a).

3. The method of claim 2, wherein said bus command includes a starting address and wherein said bus command and said reservation command together define an address range.

4. The method of claim 3, wherein said reservation command includes a count.

5. The method of claim 3, wherein if said bus command is a read operation the method further comprises the steps of:

(d) fetching data according to the read operation, the data being passed through the reserved buffer;

(e) prefetching data according to the read operation and said address range, the prefetched data being written into the reserved data buffer; and

(f) providing the fetched and prefetched data to the bus master.

6. The method of claim 5, further comprising the steps of:

(g) repeating steps (e) and (f) until said address range is exhausted; and

(h) canceling the data buffer reservation after step (g).

7. The method of claim 1, wherein said reservation command is received after a frame signal is asserted.

8. The method of claim 1, wherein the bus bridge and the bus master are for coupling to a peripheral component interconnect (PCI) bus.

9. A method of communicating between a first and a second bus bridge, the second bus bridge for coupling between the first bus bridge and a bus master, the method comprising the steps of:

(a) receiving a reservation command at the second bus bridge from the bus master; and

(b) forwarding said reservation command to the first bus bridge.

10. The method of claim 9, wherein the second bus bridge includes a programmable forwarding bit to select whether reservation commands are forwarded, the method further comprising the step of:

(c) determining whether the forwarding bit is set to enable forwarding of reservation commands before step (b), and

wherein step (b) is conditioned on the forwarding bit being set to enable forwarding.

11. The method of claim 9, the method further comprising the steps of:

(d) receiving a bus command after step (a); and

(e) reserving a data buffer for the bus master in response to said reservation command and said bus command.

12. The method of claim 11, wherein said bus command includes a starting address and wherein said bus command and said reservation command together define an address range.

13. The method of claim 12, wherein said reservation command includes a count.

14. The method of claim 12, wherein if said bus command is a read operation the method further comprises the steps of:

(f) fetching data according to the read operation, the data being passed through the reserved buffer;

(g) prefetching data according to the read operation and said address range, the prefetched data being written into the reserved data buffer; and

(h) providing the fetched and prefetched data to the bus master.

15. The method of claim 14, further comprising the steps of:

(i) repeating steps (g) and (h) until said address range is exhausted; and

(j) canceling the data buffer reservation after step (i).

16. The method of claim 9, wherein said reservation command is received after a frame signal is asserted.

17. The method of claim 9, wherein the bus bridge and the bus master are for coupling to a peripheral component interconnect (PCI) bus.

18. A method of reserving a buffer in a bus bridge, the method comprising the steps of:

(a) providing a frame indication;

(b) providing a reservation command when said frame indication is provided; and

(c) providing a bus command after step (b) and while said frame indication is provided.

19. The method of claim 18, wherein said bus command includes a starting address and wherein said bus command and said reservation command together define an address range.

20. The method of claim 18, wherein said reservation command includes a count.

21. A computer system, comprising:

a main memory;

a processor coupled to said main memory;

a mass storage system;

a bus master coupled to said mass storage system, said bus master operable to provide a bus operation including a reservation command and a bus command; and

a bus bridge coupled between said main memory and said bus master, said bus bridge having at least one assignable data buffer, said bus bridge operable to receive the bus operation and reserve a data buffer for the exclusive use of said bus master in response to said bus operation.

22. The computer system of claim 21, wherein said bus command includes a starting address and wherein said bus command and said reservation command together define an address range.

23. The computer system of claim 22, wherein said reservation command includes a count.

24. The computer system of claim 22, wherein if said bus command is a read operation

said bus bridge fetches data according to the read operation and prefetches data according to said address range, the data being provided to the bus master when requested.

25. The computer system of claim 24, wherein when said bus master reads at a last address of the address range, said bus bridge cancels the data buffer reservation.

26. The computer system of claim 21, wherein said reservation command is received after a frame signal is asserted.

27. The computer system of claim 21, wherein the bus bridge and the bus master are coupled to a peripheral component interconnect (PCI) bus.

28. The computer system of claim 21, wherein said bus operation includes a memory read operation for writing to said mass storage system.

29. A computer system, comprising:

a main memory;

a processor coupled to said main memory;

a mass storage system;

a bus master coupled to said mass storage system, said bus master operable to provide a bus operation including a reservation command and a bus command;

a first bus bridge coupled to said main memory; and

a second bus bridge coupled between said first bus bridge and said bus master, said second bus bridge having at least one assignable data buffer and a forwarding bit, said second bus bridge operable to receive the bus operation and reserve a data buffer for the exclusive use of said bus master in response to said bus operation, said second bridge operable to forward said reservation command to the first bridge if said forwarding bit is set.

30. The computer system of claim 29, wherein said bus command includes a starting address and wherein said bus command and said reservation command together define an address range.

31. The computer system of claim 30, wherein said reservation command includes a count.

32. The computer system of claim 29, wherein if said bus command is a read operation

said second bus bridge fetches data according to the read operation and prefetches data according to said address range, the data being provided to said bus master when requested.

33. The computer system of claim 32, wherein when said bus master reads at a last address of the address range, the second bus bridge cancels the data buffer reservation.

34. The computer system of claim 29, wherein said reservation command is received after a frame signal is asserted.

35. The computer system of claim 29, wherein said second bus bridge and said bus master are coupled to a peripheral component interconnect (PCI) bus.

36. The computer system of claim 29, wherein said bus operation includes a memory read operation for writing to said mass storage system.

3. Detailed Description of Invention

Field of invention:

The invention relates to a method for translating operations from one bus to another bus and more particularly to a buffer reservation mechanism for optimally transferring data between buses.

Prior art:

Personal computers are constantly changing as new technologies evolve and are incorporated into the computer. Performance improvements in the microprocessor and memory have resulted in computers so powerful that they are now capable of performing tasks that before could only be performed by large mainframe computers. However, to fully replace a mainframe computer, the computer must have significant memory and storage capacity supported by a hearty I/O (input/output) subsystem.

Several standardized I/O buses are available to the system designer including: ISA (Industry Standard Architecture); EISA (Extended Industry Standard Architecture); and PCI (Peripheral Component Interface). Today's computers are typically designed with some combination of the three. For moving data between the buses, a bridge device is typically provided.

The bridge device connects to both buses for transferring data between the buses and translating the bus control signals. The buses can be different, or especially in the case of

PCI, the bridge can simply provide an electrical extension to the same logical bus. This electrical separation makes it possible to meet the PCI bus requirement of limiting the number of physical devices on one bus segment, while at the same time not limiting the total number of PCI agents. In PCI bus vernacular, an agent is the term denoting the class of devices connecting to the bus, including master and slave devices.

These buses all support a scheme called bus mastering wherein a device or agent, usually other than the processor, may request an arbiter for control of the bus. If the arbiter grants the agent control, the agent becomes a bus master. The bus master directs its operations to another agent called a slave. The bus master may then perform operations without processor intervention much more efficiently than if the processor were involved. Many times a master on an origination bus will communicate with a slave on a destination bus. If the buses are coupled by a bridge, the performance of the bridge greatly effects the performance of the communication between the master and slave. Thus, it is desirable to optimize this pathway.

In the case of the PCI bus, one method of improving performance is to permit delayed transactions or read posting. More details on the PCI bus and on Delayed Transactions are found in the PCI Local Bus Specification version 2.1 which can be obtained from the PCI Special Interest Group, Hillsboro, Oregon. The PCI Local Bus Specification and its related documentation are hereby incorporated by reference. Delayed transactions permit the bus to be used while a slow device is preparing data in response to a request. Thus, instead of the slow device applying wait states to the bus, the bus may be used for other requests. For a bridge, the destination bus means the interface that was not acting as the target of the original request. A delayed transaction progresses to completion in three phases: the request by the master; completion of the request by the target; and the completion of the transaction by the master.

In the first phase, the master generates a transaction on the bus, the target decodes the access, latches the information required to complete the access and terminates the request with a retry-termination. Since the master cannot distinguish between a target which is completing the transaction using delayed transaction termination and a target which simply cannot complete the transaction at the current time, it must reissue the request. During the second phase, the target independently completes the request on the destination bus using the latched information from the delayed request. If the delayed request is a read, the target obtains the requested data and completion status. If the delayed request is a write, the target

delivers the write data and obtains the completion status. During the third phase, the master successfully rearbitrates for the bus and reissues the original request. The target decodes the request and provides the master with the completion status (and data if a read request). All bus commands that must complete on the destination bus before completing on the originating bus may be completed as a delayed transaction. These include interrupt acknowledge, I/O read, I/O write, configuration read, configuration write, memory read, memory read line and memory read multiple commands. Memory write and memory write and invalidate commands can complete on the originating bus before completing on the destination bus. These commands are not completed using delayed transactions termination and are normally posted.

One such bridge device is the Intel PCI to EISA bridge chip set. The 82375EB/SB PCI-EISA Bridge and the 82374EB/SB EISA system component work in tandem to provide an EISA I/O interface for computers having a PCI bus. The chip set can be either a master or slave on both the PCI and EISA buses. For PCI to EISA data transfers, four 32-bit posted write buffers are provided to enhance single cycle PCI bus transactions. For EISA to PCI data transfers, four 16-byte line buffers are included to support EISA bursting.

In order to use both buses efficiently, most bridges implement some amount of data buffering within the bridge itself. This allows the bridge to de-couple the buses from each other and let each bus run at its maximum speed without being slowed down by the other. There are generally two types of buffers that may be implemented in a bridge: write posting buffers and read prefetch or read ahead buffers. Both types can be implemented on either bus.

Write posting buffers accept write data from one bus and acknowledge reception to that bus. This frees the bus to perform other transactions. The bridge temporarily stores, or posts, the write data until it can be written to the other bus. Read prefetch buffers take the address from a single read access and read additional data speculating that it will also be needed. The bridge then holds that data in a buffer until it is either unusable or it is used by a read access.

In the Intel chip set, the buffer permits the bridge to receive short bursts of data at peak data transfer rates. For example, if a EISA device requests data from memory on the PCI bus, the bridge can burst four 32-bit data words from the PCI memory into its buffer and then release the PCI bus to other PCI requestors while the EISA device reads the buffers. Therefore, the PCI bus is not held up by the EISA device. While the buffer is filled, the

EISA device is notified to read the data from the buffer. The EISA device may then read the data from the buffer at its burst transfer speed. Thereafter, the bridge attempts to keep up with the EISA bus transfer rate by performing short burst of data across the PCI bus to keep the buffer filled. However, if another EISA device requests data from memory, the buffer must be flushed and filled again.

Buffers also introduce problems with data consistency. While data is buffered in the bridge, a bus agent and the processor may have different ideas about what is really in memory. When a bus master issues a read request through the bridge to a target memory range on the other side of the bridge, the bridge must balance between two conflicting goals: performance and data integrity. If the memory operation and transfer were optimized for performance, the bridge would preferably hold large blocks of prefetched data. However, if data is prefetched in large blocks but unused, the bridge must guard against providing stale data on a subsequent read request by a master. Additionally, prefetching large blocks of unused data hurts performance by wasting bus bandwidth. Thus, it is desirable to find a solution which both meets performance concerns and also guarantees data integrity.

Buses can also operate at different frequencies. The bridge must therefore synchronize the signals as they are translated from one bus at one frequency to another bus at another frequency. If a continuous flow of data is to be achieved, the synchronization must not be at the expense of performance.

Means to solve problems:

A bus bridge for connecting two buses includes a plurality of data buffers. The data buffers are used for storing read data, storing prefetched or read-ahead data or for storing write posted data. According to the invention, the use of these data buffers reduces the number of times a device will access the buses, thereby reducing bus latency.

The bridge further includes a mechanism for reserving a data buffer. Two alternative mechanisms are disclosed whereby a bus master or agent coupled to the bridge may request that the bridge reserve a buffer for the bus master=s intended data read transfer. If a hierarchy of bridges is present, the reservation request is passed on to upstream bridges supporting the buffer reservation mechanism.

The reservation request received by the bridge includes the addressing information necessary for the bridge to read-ahead or prefetch data as the bus master removes data from the reserved data buffer. When a reservation is placed, three 128-byte data buffers are reserved for the bus master. Data is prefetched into these data buffers in order and on a

rotating basis as read by the master. When a data buffer is first accessed by the master, a next data buffer is prefetched. When the complete address range has been read and the data has been removed by the bus master, the reservation is canceled. If the buffer has not been accessed by a bus master within a programmable period of time, the reservation is also canceled. A reservation algorithm is disclosed for managing the reserved buffers.

The bridge further includes two alternative algorithms for managing the buffers if the reservation mechanism is not available to an agent. With the default algorithm, when a new bus master attempts a read without first reserving a buffer, the bridge allocates a buffer, comprising three 128-byte data buffers, to this master if a buffer is available, otherwise the master is forced to retry the read operation. Once the buffer is allocated, the bridge reads the requested data into a first of the 128-byte data buffers and subsequently provides the data to the bus master. If the master burst reads past the end of the first data buffer and the next read operation from the master starts where the last read ended, then prefetching is enabled for this master. On the next sequential read the bridge will again read the requested data into a second data buffer and subsequently provide it to the master. At this point prefetching is already enabled. Whenever the master again burst reads past the end of one of the buffers, the bridge automatically prefetches data for the next 128-byte buffer. The three 128-byte data buffers are filled in order and on a rotating basis. If the bus master requests a non-sequential read or if the CPU performs a write transaction which might affect the master, the buffer is flushed.

A conservative algorithm is similar to the default algorithm, but the bridge does no prefetching. If the bus master does not read all of the data, the remaining data is flushed when the master disconnects from the bus.

Embodiment:

Referring now to Figure 1A, the bridge device 106 is illustrated according to the preferred embodiment. The bridge device 106 couples a primary bus 110 to a secondary bus 112 of a computer system 100. The computer system 100 further includes a CPU 102 and memory 104 coupled to the primary bus 110. In the preferred embodiment, the CPU 102 includes a processor (not shown) such as a 486, Pentium7 or 586 class processor by Intel or their equivalents and standard computer peripherals and support logic (not shown) as is common in IBM compatible PCs. It is understood that other processors and peripherals could of course be utilized.

A bus master 108, such as a hard disk controller, video graphics controller or network interface controller, and bus slave 114, such as a serial port or parallel port are further connected to the secondary bus 112. For purposes of the principles disclosed in this invention, it is not necessary to identify the specific functionality of the bus master 108 and slave 114, therefore, they are simply referred to herein as generic bus masters and slaves.

The primary and secondary buses 110 and 112 are standard input/output buses such as the peripheral component interconnect (PCI) bus. It is understood that the principles herein disclosed may be applied to other buses as well, such as the industry standard architecture (ISA), extended industry standard architecture (EISA) and microchannel architecture (MCA) bus. Further, as can be appreciated, the same bridge design principles may also be applied to couple two dissimilar buses, such as a PCI bus and an EISA bus. In the preferred embodiment, the primary bus is a local PCI bus with limited electrical fanout and the bridge 106 is used to extend the functionality of the primary bus 110 by way of the secondary PCI bus 112.

In an alternative to Figure 1A, shown in Figure 1B, the secondary bus 112 is in a remote location from the computer system 100. The bridge 106 is further comprised of an upstream part 106a and a downstream part 106b. A cable 118 couples the upstream and downstream bridge instances together. The downstream part 106b is coupled to the secondary bus 112 for communicating with a plurality of bus masters 108a-d. Each bus master is further coupled to a plurality of hard disk drives 116a-d for providing mass storage to the computer system 100. In both Figures 1A and 1B, the bridge 106 provides a common interface for devices on the secondary bus, such as the bus master 108, to communicate with devices on the primary bus, such as the memory 104. The bus master 108 may also communicate with other secondary bus devices, such as the bus slave 114, without requiring any bandwidth from the primary bus 110. Thus, in addition to being a translation unit and bus extender, the bridge 106 also functions as a isolation buffer.

When large blocks of data are transferred through the bridge, such as when the bus master 108 reads or writes to the memory 104, the bridge provides multiple buffers for read prefetching and write posting to enhance the overall system performance. This helps eliminate the typical requirement that both buses be communicating with the bridge 106 before a transfer can take place. The read prefetching is managed according to several alternative algorithms. One algorithm includes a mechanism in which a bus master can reserve a read prefetch buffer for very efficient bus transfers. Other adaptive algorithms handle those cases where the bus master does not request a buffer reservation. However, even for these cases, the algorithms attempt to reserve a buffer and prefetch data when certain conditions signify that a read prefetch will have a high probability of success.

Now turning to Figures 2A and 2B, a block diagram of first and second embodiments of the bridge device 106 are illustrated. As can be seen, the primary bus couples to a primary

bus interface 200 and the secondary bus couples to a secondary bus interface 202. The bridge 106 contains a buffer region 204 for receiving incoming data, read prefetching or read-ahead, or for write posting. The buffers are configured according to the type of operation, as described below.

Specifically, in the bridge 106, four 128-byte write posting buffers 218 are provided to enhance the performance of write operations. When the bus master 108 performs a write operation to main memory 104, the data is written or posted into the write buffers 218 and the bus master 108 receives a completion indication from the bridge 106 before the data is actually written to main memory 104. The bridge 106 then subsequently completes the write operation to main memory 104 without further delay. Thus, the time the bus master 108 controls the secondary bus 112 is greatly reduced since it does not wait for the actual completion indication.

The buffer region 204 also includes eight default buffers 210 for enhancing bus master read operations from main memory 104. A default buffer management algorithm manages the default buffers 210 for storing read data and prefetching read data under certain conditions. The default buffer management algorithm balances between holding the prefetch data as long as possible and yet avoids the possibility of retaining stale data. The buffers are filled and emptied using three 3-bit pointers to 32-bit double word chunks of data (hereinafter called DWORDS) in the buffers. A buffer input pointer (BUFF_IN_PNTR) points to where the next Dword will be stored; a buffer output pointer (BUFF_OUT_PNTR) points to where the next Dword will be read from; and a buffer valid pointer (BUFF_VALID_PNTR) indicates which addresses between BUFF_IN_PNTR and BUFF_OUT_PNTR are valid. Only certain addresses may be valid since to prevent bus locked situations a read completed after a posted memory write can be stored in the buffer, but cannot become valid until after the posted memory write completes.

As an alternative to the default buffer management algorithm, a conservative algorithm is provided to manage the default buffers 210. The conservative algorithm never prefetches and generally only reads data that has been requested by a bus master. Thus, stale data is avoided altogether.

The buffer region 204 also includes a pool of eight reservation resources 220 for enhancing bus master read operations from main memory 104. These resources 220 are managed by a reservation algorithm, discussed below. Each reservation resource 206 consists of three 128-byte data buffers and a corresponding reservation information block

208. Each information block 208 includes the registers necessary to control the reservation buffers, which include: a reservation base address register (RSRV_BASE_ADDR) containing a 64-bit starting address of the reservation; a reservation limit address register (RSRV_LIMIT_ADDR) containing a 64-bit address one larger than the last address of the reservation; a discard timer value programmed to one of four values (off, 25ms 50ms, or 100ms) for automatically canceling inactive reservation resources; a last buffer indicator register (LAST_BUFF) containing a 5 bit pointer to the last line buffer data was read into; a buffer start pointer register (BUFF_START_PNTR) containing a 64-bit starting address of each line buffer; a buffer limit pointer register (BUFF_LIMIT_PNTR) containing a 64-bit address one larger than the last address of the line buffer; a buffer input pointer register (BUFF_IN_PNTR) for holding the address where the next Dword is stored; a buffer output pointer register (BUFF_OUT_PNTR) for holding the address where the next Dword will be read from; a buffer valid pointer register (BUFF_VALID_PNTR) for indicating which addresses between the BUFF_IN_PNTR and BUFF_OUT_PNTR are valid; a buffer state register; and a buffer prefetch flag register.

The default buffers 210 and reservation resources 220 may be allocated to each requesting master or to each process or thread, thereby allowing a single master to have multiple buffers. The buffer region 204 is organized as a FIFO, and of course, may be sized according to the requirements of the master for optimal bus utilization. Since the bridge 106 is preferably transparent to the requestor and its target, it is desirable for the buffer region 204 to also be transparent to the transaction. However, to optimize the operation of the buffer region 204 it is desirable to control the buffers so that data will be prefetched efficiently and the prefetched data will be coherent with data in main memory. Therefore, to optimize the buffers for the pending transactions, it is desirable for the bridge to receive the range of addresses the master intends to read. Thus, once a bridge obtains the address information, the bridge may freely prefetch data up to the end of the address range. Two alternatives are provided for requesting a reservation resource.

In a first alternative reservation mechanism, illustrated in Figure 3A, the secondary interface 202 contains a command decode block 212a. It is noted that while the command decode block 212a is located in the secondary interface 202 because memory read requests are from downstream bus masters, it is contemplated that a command decode block could also be effective in the primary interface 200 for read operations in the opposite direction. The reservation resource is most effective for read operations initiated by a bus master

residing on the secondary bus 112 from main memory 104, thus, the following description is presented in that context.

A reservation resource 220 can be reserved by a bus master 108 when a special bus command is received by the bridge 106. At this point, the reader may also want to refer to Figure 7 4A. When the bus master 108 desires to reserve a reservation resource 220 for read transactions from the target memory 104, the bus master 108 issues a first read transaction containing a reservation command (RSRV_COMMAND) for indicating a total number of bytes to be read from the target memory 104. The RSRV_COMMAND is a special PCI bus command represented by the command/byte enable (C/BE#) signals on the PCI bus and is presented to the bridge 106 on the first read transaction when a frame (FRAME#) signal is asserted to indicate the beginning of a transaction, as shown at clock 2 of Figure 7 4A. The reservation command also includes a count on the AD[31:0] lines to indicate the number of bytes desired. From clock 3 forward, the PCI transaction continues as normal: the type of PCI transaction is provided on the C/BE# lines along with a starting address on the address/data (AD) lines; and data is transferred when available (as shown in clocks 5, 7 and 9). Thus, the count (clock 2) and starting address (clock 3) indicate a range of memory to read. It is understood that there are alternative ways to indicate this range for reserving a buffer, such as by indicating a starting address and ending address. These alternatives are contemplated for the reservation command.

For 64 bit addressing, illustrated in Figure 8 4B, the RSRV_COMMAND precedes the dual address command and actual read command. More details on the dual address command is found in the PCI Specification referred to above.

When the bridge 106 receives the RSVR_COMMAND, the registers in the reservation information block 208 are initialized. The starting address is stored in the RSRV_BASE_ADDR, the ending address is determined by adding the starting address to the total byte count, which is incremented by one and stored in the RSRV_LIMIT_ADDR. One of the three 128-byte buffers is initialized by storing the starting address in the BUFF_START_PNTR and the buffer ending address in the BUFF_LIMIT_PNTR. Also, the buffer state and prefetch flag for each of the three buffers is initialized.

Once the reservation is placed, the bus master 108 may disconnect from the bridge 106 and reconnect later to resume the operation without reissuing the RSVR_COMMAND. Thus, in this alternative, a single PCI bus command indicates the number of bytes to be

transferred. The penalty for this additional information is only one clock cycle, since once the buffer is reserved, the data transactions proceed as normal.

The target does not receive the RSRV_COMMAND. However, intermediate bridges may receive the RSRV_COMMAND from downstream bridges. If a hierarchy of bridges is present, as illustrated in Figures 5A and 6B, each bridge supporting the RSRV_COMMAND may pass the reservation upstream. In the configuration of each bridge, a configuration bit 214 (Figure 3A) is cleared if the upstream bridge does not support buffer reservations, otherwise the bit is set by configuration software during computer initialization operations and the reservation will be passed upstream. Figure 5A shows an example hierarchy of bridges where a bridge 300 is a non-reservation bridge sandwiched between two bridges 106a and 106b which support buffer reservations. In this example, since the bridge 300 does not support buffer reservations, the configuration bit in bridge 106b would be cleared so that the RSRV_COMMAND would not be forwarded to the non-reservation bridge 300. Thus, even though bridge 106a supports buffer reservations, this feature is not utilized in bridge 106a.

Figure 6B illustrates an example hierarchy of bridges in which a top-most bridge 300 does not support buffer reservations, but a lower two bridges 106a and 106b do support it. In this example, since the bridge 300 does not support buffer reservations, the configuration bit in the bridge closest to the non-reservation bridge, bridge 106a, would be cleared so that the RSRV_COMMAND would not be forwarded to the non-reservation bridge 300. The configuration bit 214 in bridge 106b is set so that the RSRV_COMMAND received by bridge 106b is forwarded to bridge 106a. Thus, a non-reservation bridge acts to cut off the benefits of buffer reservations when implemented with the RSRV_COMMAND.

Referring now to a second alternative reservation mechanism, illustrated in Figure 4B, the bridge 106 contains an address decode block 212b for decoding operations to a special address. In the memory address map of the computer system 100, this special memory address is allocated for this reservation mechanism. Write operations to this special address are interpreted as reservation operations by the bridge 106. When the bus master 108 desires a buffer reservation for a read operation, it first executes a 2-3 32-bit (Dword) burst write to the special address. The bridge 106 intercepts the write operation and stores the command in a reservation mailbox 216. The first Dword is the total number of bytes the bus master 108 intends to read. The second Dword is the least significant 32 bits of the starting address. If 64-bit addressing is utilized, then a third Dword is written containing the

most significant 32 bits of the starting memory address. If only two Dwords are written, the bridge assumes the most significant 32 bits are zero. The bridge 106 intercepts and claims the write operation to the mailbox register 216 by asserting a device select (DEVSEL#) signal since the address is not assigned to any devices on the secondary bus 112. When the bridge 106 receives the transaction, it reserves a buffer if one is available, and stores the address, length and other information in the mailbox 216 for forwarding to other intermediate bridges, if necessary. When the reservation is placed, the reservation information block is also initialized as described above. Thus, in this alternative, a write to a special address indicates the starting address and number of bytes to be transferred. This information is utilized by the bridge 106 to prefetch data for the requesting master 108.

If a hierarchy of bridges is present as illustrated in Figures 5A and 6B, each bridge supporting buffer reservations may pass the reservation information upstream. In the configuration of each bridge, the topmost bridge in the hierarchy is disabled from forwarding the reservation. Figure 7A illustrates a hierarchy of bridges where a bridge 300 is a non-reservation bridge sandwiched between two bridges 106a and 106b which support buffer reservations. In this example, since write transactions to global memory address are used to pass the reservation information between the bridges, the configuration bit and bridge 106b would be set so that the reservation would be forwarded to the non-reservation bridge 300 and passed to the reservation bridge 106b. The non-reservation bridge 300 does not recognize the difference between a normal memory transaction and a reservation write transaction. If the bridge 106a was coupled to the memory 104 on its primary side, the configuration bit and bridge 106a would be set to disable forwarding of the reservation information to memory. Thus, each bridge capable of supporting buffer reservations receives the reservation information.

Figure 6B illustrates a hierarchy of bridges in which a topmost bridge 300 does not support buffer reservations, but a lower two bridges 106a and 106b do support it. In this example, since the bridge 300 is the topmost bridge and because it does not support buffer reservations, the configuration bit in bridge 106a would be cleared to disable forwarding of the reservation information. The reservation is passed between the bridges 106a and 106b. Thus, if a non-reservation bridge is located at the top level, the reservation bridge below the non-reservation bridge is disabled from forwarding the reservation. One advantage of this method over the first alternative is that script-driven bus masters can readily implement the memory write transaction with little modification.

Both alternatives allow the bridge 106 to prefetch data in longer, more efficient transactions. By utilizing the buffer reservation, unused prefetched data is avoided since only the data requested is prefetched. Coherency problems between the memory 104 and data stored in the buffers 206 and 210 are also avoided since only requested data is prefetched and it can be assumed that no bus master would use this data until the requestor completes the transfer. Thus, data prefetched according to the preferred embodiment can be held and considered valid until the transfer completes.

Read operations may be performed by the bridge 106 with a technique known as delayed transactions. More details on delayed transactions are found in the PCI Specification incorporated above and in the background section of this specification. Briefly though, delayed transactions are a method of termination and retry used by PCI bridges and PCI I/O controllers which cannot complete a data transaction within a certain specified time. Instead of a first requesting bus master waiting for a slow I/O controller or bridge to respond, the I/O controller or bridge may force the first requesting bus master off the bus so that other bus masters are allowed to use the bus bandwidth that would normally be wasted holding the first bus master in wait states. A delayed transaction normally progresses to completion in three phases: a request by a bus master; completion of the request by a target; and a completion of the transaction by the bus master.

In the nomenclature of the PCI bus, a delayed read request is a transaction that must complete on the destination bus before completing on the originating bus and can be an I/O read, configuration read, memory read, memory read line, or memory read multiple command. Referring to Figure 1A, if the bus master 108 requests data from memory 104, the secondary bus 112 is the originating bus and the primary bus 110 is the destination bus. Once a request has been attempted on the destination bus, it must continue to be repeated until it completes on the destination bus. Until that time, the delayed read request is only a request and may be discarded at any time to prevent bus deadlock or improved performance since the bus master 108 must repeat the request later. A delayed read completion is a read transaction which has completed on the destination bus and is now moving toward the originating bus to complete. The delayed read completion contains the data requested by the bus master 108 and the status of the target (memory 104). Once the delayed read request is executed on the destination bus it becomes a delayed read completion.

An example of a simple delayed transaction without the buffer reservation mechanisms is helpful to illustrate its effect on the buffers region 204. It is noted that

delayed transactions may be used with or without the buffer reservation mechanisms herein disclosed. Also, the buffer reservation mechanisms may be used with or without delayed transactions, but is it assumed they will be used together for maximum performance.

Figure 14 is used as an example. During a first phase, the bus master 108 generates a memory read transaction on the secondary bus 112. The bridge 106 decodes the transaction and, recognizing that the main memory is on the primary bus, claims the transaction by asserting the DEVSEL# signal, latches the transaction information required to complete the access, and terminates the request using conventional PCI retry techniques. The latched request information is referred to as a delayed request, or more specifically in this example, a delayed read request. According to PCI convention, the bus master 108 must repeat the transaction. Meanwhile, using the latched transaction information, the bridge 106 generates a memory read transaction on the primary bus 110. If the target (main memory 104) is not ready, it can also force another delayed transaction by issuing a retry to the bridge 106. However, assuming the memory 104 does not issue a retry, the memory 104 processes the request and the bridge 106 receives the completion status and the requested data into one of its buffers 204. If the delayed request was a write operation, the bridge 106 would obtain the completion status from the main memory 104. The result of completing the delayed request on the destination bus (primary bus 110) produces a delayed completion, which consists of the latched information of the delayed request and the completion status and data. The bridge 106 stores the data and completion status in the buffer region 204 until the bus master 108 repeats the initial request.

During the third phase, the bus master successfully rearbitrates for the bus and reissues the original memory read request. The bridge 106 decodes the request, claims the operation and provides the data to the bus master 108. At this point, the delayed completion is retired and the transaction has completed. The status returned to the bus master 108 is exactly the same as the bridge 106 obtained from the target (memory 104) when it executed the read transaction, i.e., master-abort, target-abort, parity error, normal, or disconnect.

By using the delayed transaction technique in combination with the buffer reservation mechanisms herein disclosed, read operations between bus masters and bus slaves can be performed more efficiently than before.

Managing the buffer region 204 is desirable to obtaining efficient data transfers. The basic component of the buffer region 204 is the 128-byte buffer, referred to generically as a buffer 500 in Figure 5A. Read prefetch data and posted write data may both be stored in

the bridge 106 (in different buffers), but the read prefetch data cannot become valid until after the posted memory write completes. As data is written into the buffer 500, a BUFF_IN_PNTR is incremented to indicate the next address. Data is emptied from the buffer 500 beginning at the address indicated by BUFF_OUT_PNTR. The BUFF_VALID_PNTR indicates which region of the buffer 500 is valid. Addresses between BUFF_OUT_PNTR and BUFF_VALID_PNTR are valid addresses. When a delayed read request is started, the BUFF_IN_PNTR and BUFF_OUT_PNTR are initialized to bits 4-2 of the PCI transaction address. As each Dword of data arrives, it is stored at the address indicated by the BUFF_IN_PNTR and the pointer is then incremented. If there are no preceding unexecuted posted memory write addressing targets on the destination bus, the BUFF_VALID pointer can be incremented when BUFF_IN_PNTR is incremented. If there is a preceding posted memory write, then BUFF_VALID_PNTR is updated after the write completes. As the bus master 108 takes data from the buffer, BUFF_OUT_PNTR is incremented after each Dword is removed.

Figure 9B illustrates how the reservation resource buffer 206 is constructed. The reservation buffer 206 has a beginning address, as indicated by RSRV_BASE_ADDR and ending address, as indicated by RSRV_LIMIT_ADDR. The reservation buffer 206 is comprised of three 128-byte data buffers (of the type shown in Figure 9A). As a read transaction progresses, these three buffers are reused so as to simulate one long buffer of length RSRV_LIMIT_ADDR minus RSRV_BASE_ADDR. Each individual buffer has a starting address, as indicated by BUFF_START, and an ending address as indicated by BUFF_LIMIT.

The state of the buffer is indicated by two variables: BUFF_STATE and PREFETCH_FLAG. The PREFETCH_FLAG is used to distinguish between a prefetch operation and an actual read request. The state of the data stored in the buffers is illustrated in figure 10A. The state of the buffer is designated as EMPTY after a system reset, a buffer timeout or buffer flush. When the buffer is designated EMPTY it is available for allocation. If the buffer is servicing a read request or is performing a read prefetch, the buffer is designated as REQUEST. Once some data begins to arrive into the buffer, the state transitions from REQUEST to PART_COMP. Once all the data has been received, the state of the buffer transitions from PART_COMP to COMPLETE. If the buffer's state is REQUEST, PART_COMP or COMPLETE and the buffer is flushed, the state returns to

EMPTY. If the buffer is designated as COMPLETE and the bus master reads all of the data from the buffer, the state is changed from COMPLETE to EMPTY.

For the reservation algorithm, the buffer is only flushed when the last DWORD of data is removed from the buffer. However, flushing of prefetch data is done from time to time with the default or conservative algorithms to guarantee that a bus master is never given stale data. For this reason, it is important to track the state of the buffers.

Prefetch data refers to the remnant of a buffer which was filled with a real delayed read request from a master, but when the master reconnected only part of the data was taken before the master disconnected. A master's prefetch data is flushed under the following conditions: (1) the master attempts a read from an address which is not sequential from that master's last read; (2) the CPU 102 does a write transaction which might affect this master such as an I/O write to a secondary bus target, a memory write to the general secondary memory range register, or a memory write to the slot-specific memory range register for this master; and (3) a secondary bus master does a write transaction which might affect this master such as an I/O write to a secondary bus target, a memory write to the general secondary memory range register, a memory write to the slot-specific memory range register for this master, or a CPU 102 write to any configuration register.

The states for the PREFETCH_FLAG are described with reference to Figure 11. When the BUFF_STATE is designated REQUEST, PART_COMP or COMPLETE, the PREFETCH_FLAG indicates whether the read is a prefetch or an actual REQUEST. When the PREFETCH_FLAG is set, a prefetch operation is in progress, while when the PREFETCH_FLAG is cleared a real REQUEST is in progress. The transitions from the set and clear states are indicated in the following equations:

```
A =   (prefetch) OR
      (after master read AND BUFF_STATE=(PART_COMP OR COMPLETE));
B =   (flush AND BUFF_STATE=(REQUEST, PART_COMP OR COMPLETE));
C =   (master read transaction AND delayed read request) OR
      (prefetch complete AND BUFF_STATE=EMPTY) OR
      (flush AND BUFF_STATE=COMPLETE) OR
      (real delayed read request);
D =   (master read transaction AND delayed read request) OR
      (prefetch complete AND BUFF_STATE=EMPTY);
```

A buffer 204 is only available for a new REQUEST if it is EMPTY or it contains prefetch data and there is not already a prefetch in progress for that buffer. When a bus master 108 attempts a read and the bridge 106 initiates a delayed read request, BUFF_STATE is changed to REQUEST, and PREFETCH_FLAG is cleared. If the read operation is initiated by a bridge 106 prefetch, then BUFF_STATE is changed to REQUEST, and PREFETCH_FLAG is set.

When the first data arrives BUFF_STATE is changed to PART_COMP. If the master repeats the read transaction and the state is PART_COMP or COMPLETE, read data is provided to the bus master 108. If the bus master 108 does not take all of the data before disconnecting from the bridge 106, PREFETCH_FLAG is set to indicate that the remainder of the data in the buffer 204 is now considered to be prefetched data, but the state remains unchanged. If the bus master 108 takes the last data, i.e., (BUFF_OUT_PNTR > = BUFF_IN_PNTR > after the read) when the state is COMPLETE, the state is changed to EMPTY.

If a flush event occurs for the buffer 204 and its PREFETCH_FLAG is set, and the state is designated REQUEST, PART_COMP or EMPTY, a prefetch is currently in progress, so the state is changed to EMPTY, but the PREFETCH_FLAG remains set until the pending prefetch is complete. The buffer 204 cannot be reused for another prefetch until the PREFETCH_FLAG is cleared. If a flush event occurs for the buffer 204 and its PREFETCH_FLAG is set, and the state is designated COMPLETE, the state is changed to EMPTY and the PREFETCH_FLAG is cleared.

If a real delayed requested needs the buffer 204 while the PREFETCH_FLAG is set, the new request is given the buffer. The PREFETCH_FLAG is cleared (to cause incoming prefetch data to be discarded) and the state is changed to REQUEST.

When prefetch data arrives for the buffer 204 and PREFETCH_FLAG is cleared, the data is discarded (the buffer has been reused for a real request). When prefetch data arrives for a buffer 204 and the state is EMPTY, the data is discarded (the buffer is flushed). When the last prefetch data arrives, the PREFETCH_FLAG is cleared and the buffer 204 is available to be reused.

Three prefetch and flush algorithms are implemented in the bridge 106: default algorithm, conservative algorithm, and reservation algorithm. The default algorithm should be applicable to most bus masters. The conservative algorithm prefetches less and flushes more, but is implemented only in the case where the default algorithm does not work for a

particular bus master. The reservation algorithm is used by more sophisticated bus masters which are capable of initiating the reservation mechanism.

Now referring to Figures 12 7A through 19 7H, the prefetch and flush algorithms are described according to the preferred embodiment. It is understood that the bridge 106 performs various processes related to read and writing data, however, for simplicity these steps have been omitted. Further, although the steps relating to write posting are not shown, it is understood that the bridge performs these operations in conjunction with the read prefetch and flush processes described herein.

The prefetch and flush algorithms begin at Figure 12 7A with the receipt of a bus transaction on the secondary bus interface 202. At step 700 (Fig. 12 7A), it is determined whether a configuration command was received. If so, control proceeds to step 710 (Fig. 13 7B) to complete the command, otherwise control continues to step 702. At step 702 it is determined if a write operation has been issued to the bridge 106. If so, control proceeds to step 712 (Fig. 14 7C) where it is determined if the write operation affects one of the default buffers that have been allocated. If a bus master 108 is allocated one of the default buffers 210, and a write operation is performed to the bus master 108, such as a configuration write or command, then it is possible that a new process has been initiated, in which case any prefetched data contained in the default buffer 210 is subject to being stale or non-conforming. In such a case, the buffer is said to be affected by the write command. If such is the case, then control proceeds to step 714 where the default buffer 210 corresponding to the slot or master affected by the write operation is flushed. Control then proceeds from step 714 to step 716. If it is determined at step 712 that the write command does not affect any default buffers 210, then control proceeds from step 712 to step 716. At step 716, the remaining write operation processing is handled, including receiving any write data into the write posting buffers and updating the BUFF_VALID_PNTR (at step 718). The BUFF_VALID_PNTR is only updated for the default and conservative algorithms. The routine then terminates, returning to whatever process called it.

Returning to step 702, if it is determined that the operation is not a write operation then control proceeds to step 704 where it is determined if the operation is a reservation command. If so, control proceeds to step 720 (Fig. 15 7D) where the reservation command is received by the bridge 106. Control then proceeds to step 722 where it is determined if one of the reservation resources 220 is available for this request. If not so, then control proceeds to step 742 where a retry is issued to the requesting bus master, and the routine ends. If a

reservation resource 220 is available, then control instead proceeds from step 722 to step 724 where the reservation resource registers are initialized with data (i.e., address and count) contained in the reservation command. Control then proceeds to step 728 where three 128 byte buffers are allocated to this request and to step 730 where the status for these buffers is changed to EMPTY. Control then proceeds from step 730 to step 732 where the bridge 106 prefetches three to four lines of data into a first 128-byte buffer of buffer 206. Control then proceeds to step 734 where the last buffer pointer (LAST_BUFF_PNTR) is initialized to indicate which buffer will be written to next.

Control then proceeds to step 736 where it is determined if the forwarding configuration bit is set. If set, then control proceeds to step 738 where the reservation command is forwarded upstream to any upstream bridge. Control then proceeds from step 738 to step 740. If at step 736, the configuration bit is not set, then control proceeds directly to step 740. At step 740, a reservation timer is reset and started. The reservation timer cancels the reservation if after a certain amount of time, such as 25ms, 50ms or 100ms, the reservation resource has not been accessed. It is assumed that the data will be stale after this period of inactivity. The routine then ends.

Returning to step 704, if it was determined that the operation is not a reservation command then control proceeds to step 706 where it is determined if the operation is a read retry from a bus master. If not so, then the operation is deemed to be a first read operation and control proceeds to step 744 (Fig. 17).

At step 744 it is determined if the requesting bus master already has a default buffer 210 allocated to it or is a new bus master. If it is determined that this is not a new master then control proceeds to step 746 where it is determined if the requested address is equal to the address pointed to by the BUFF_OUT_PNTR. If so, then control proceeds to step 750. If not so, then control proceeds to step 748 where the allocated default buffer 210 is flushed because the master has begun reading from a new, not contiguous address. Control then proceeds to step 750 where the default buffer 210 is reallocated to this master and the buffer registers are initialized as described above. Control then proceeds to step 768.

If at step 744 it was determined that this is a new master, then control proceeds to step 756 where it is determined if any one of the default buffers 210 currently are designated as EMPTY. If so, then control proceeds to step 758 where one of the EMPTY default buffers 210 is allocated to the new master. From step 758, control proceeds to step 768. If not so, then control proceeds to step 760 where it is determined if any default buffers 210

are currently designated as REQUEST or PART_COMP. If so, then control proceeds to step 762 where it is determined if the PREFETCH_FLAG is set and if so, then control proceeds to step 764 where the default buffer 210 is allocated to the new master. Control then proceeds from step 764 to step 768. If at steps 760 and 762, the determinations are negative, then control proceeds to step 766 where the bus master is asked to retry the operation at a later time.

At step 768, control disables prefetching for the newly allocated or re-allocated default buffer 210. Control then proceeds to step 770 (Fig. 7E) where the read request information is stored. Control then proceeds to step 772 where the bus master is asked to retry the operation while the bridge 106 gathers the requested data from the target on the destination bus. Control then proceeds to step 774 where it is determined if the read request was a memory read multiple command. If so, then control proceeds to step 776 where the bridge 106 reads three to four lines of data from main memory 104. Control then proceeds from step 776 to step 784. If at step 774 it is determined that the read command was not a memory read multiple, then control proceeds to step 778 where it is determined if read command was a memory read line command. If so, then control proceeds to step 780 where the bridge 106 reads one line of data from main memory 104 and control then proceeds to step 784. If at step 778 if is determined that the command was not a memory read line command, then control proceeds to step 782 where the bridge 106 reads one Dword of data from main memory 104 and then control proceeds to step 784.

At step 784 the bridge 106 receives the data from main memory 104 and at step 786 the bridge 106 stores the received data in the allocated buffer and ends. The routine will later be called when the requesting bus master retries the operation.

Returning now to step 706, if it is determined that the operation is a read retry operation from a bus master then control proceeds to step 708 where it is further determined if this bus master has previously requested a buffer reservation. If so, then control proceeds to step 788 (Fig. 7G) where it is determined if the address for the retry read operation is within the limits of the reservation resource 220. If not so, then control proceeds to step 790 where the reservation is canceled and control proceeds to step 744 (Fig. 7F) to begin a read operation using the default algorithm. If at step 788 it was determined that the read operation is within the reservation limits then control proceeds to step 792 where it is next determined if the read address is within one of the three buffer limits. If so, then control proceeds to step 794 where the requested Dword of data is provided to the bus master. From

step 794, control proceeds to step 796 where it is determined if this is the first time the bus master has read from this particular 128 byte buffer.

If this is the first time, then control proceeds to step 798 where the bridge 106 prefetches data into the next buffer as indicated by the LAST_BUFF_PNTR. The buffers 206 are filled according to a least recently used algorithm. Once data is read from a buffer for the first time, the least recently used buffer is used for this next data prefetch operation. So at this point, the bus master is reading from one buffer, a next buffer holds prefetched data, and another buffer holds previously prefetched data. So even if the bus master needs to back-up or step back from it=s current address, the data will still likely be present in the previously prefetched buffer. Control proceeds from step 798 to step 800. If this is not the first time, then control proceeds from step 796 to step 800 where it is determined if the read operation by the bus master is a burst read. If so, then control proceeds back to step 788 to provide additional data. If not so, then control proceeds to the end.

If at step 792 it was determined that the read operation was not within the buffer limits then control proceeds to step 804 where the bridge 106 forces the master to retry the operation while the bridge 106 fetches the requested data. Control then proceeds from step 804 to step 806 where the bridge reads the data from main memory 104, proceeds to step 808 where it receives the data, and proceeds to step 810 where the received data is stored in the buffer 206 and the bridge 106 then awaits for the bus master to retry the read operation.

Returning to step 708, if it was determined that there was not a reservation resource previously requested for this read operation then control proceeds to the default algorithm at step 812 (Fig. 7H (19)).

At step 812, the requested data is provided to the bus master from one of the default buffers 210. Control then proceeds to step 814 where it is determined if the read operation is a burst operation. If not so, then processing continues to step 815 where it is determined if the conservative algorithm is selected. If so, then control proceeds to step 817 where the buffer is flushed. If the conservative algorithm is not selected, then control proceeds to the end. If at step 814 a burst operation is in progress, then control proceeds to step 816 to determine if the next requested address is contained within the data buffer. If so, then control proceeds to step 818 where the data is provided to the bus master. Control then proceeds back to step 814 to provide additional data if requested. If at step 816 it is determined that the next requested address is not contained within the buffer, then control proceeds to step 820 where it is determined if prefetching has been enabled for this buffer.

If not so, then control proceeds to step 832 where prefetching is enabled. Control then proceeds to step 770, as described above.

If at step 820 it is determined that prefetching has been enabled, then control proceeds to step 822 where the read transaction information is stored in the bridge 106. Control then proceeds to steps 824 and 826 where the bridge 106 issues a retry to the bus master and the bridge issues a read operation to main memory 104. From this point forward, prefetching is enabled for this bus master. At step 828 the bridge 106 receives the read data and at step 830 the bridge stores the received data into the buffer and waits for the bus master to retry the read operation.

The foregoing disclosure and description of the invention are illustrative and explanatory thereof, and various changes in the size, shape, materials, components, circuit elements, wiring connections and contacts, as well as in the details of the illustrated circuitry and construction and method of operation may be made without departing from the spirit of the invention.

4. Brief Description of Drawings

Figure 1A is a block diagram of a computer system incorporating a bridge device according to the preferred embodiment;

Figure 1B is a more detailed block diagram of a computer system incorporating a bridge device according to the preferred embodiment;

Figure 3 is a block diagram illustrating the functional blocks of the bridge device according to the preferred embodiment;

Figure 4 is a block diagram illustrating the functional blocks of the bridge device according to the preferred embodiment;

Figure 5A is a block diagram illustrating an exemplary hierarchy of buses with a non-reservation bridge coupled between two reservation bridges according to the preferred embodiment;

Figure 5B is a block diagram illustrating an exemplary hierarchy of buses with a non-reservation bridge acting as a host bridge at the host top of the hierarchy;

Figure 7A is a timing diagram illustrating a PCI bus read transaction incorporating a reservation command;

Figure 8B is a timing diagram illustrating a PCI bus read transaction with a 64-bit address incorporating a reservation command;

Figure 9 shows block diagrams of buffers, wherein A is a block diagram of a single 128-byte data buffer according to the preferred embodiment, and B is a block diagram of a reservation buffer according to the preferred embodiment;

Figure 10A is a state diagram illustrating buffer states according to the preferred embodiment;

Figure 11B is a state diagram illustrating buffer prefetch states according to the preferred embodiment; and

Figure 12 is a flow diagram illustrating a prefetch and flush algorithm according to the preferred embodiment;

Figure 13 is a flow diagram which follows "A" of Figure 12, according to the preferred embodiment;

Figure 14 is a flow diagram which follows "B" of Figure 12, according to the preferred embodiment;

Figure 15 is a flow diagram which follows "C" of Figure 12, according to the preferred embodiment;

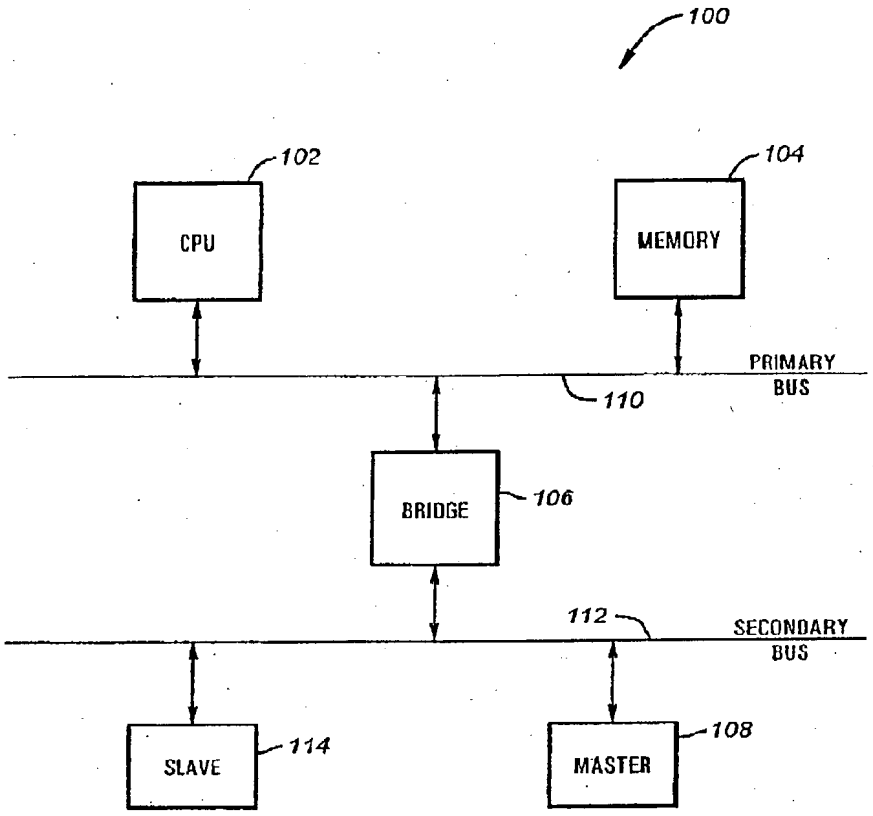
Figure 16 is a flow diagram which follows "G" of Figures 17 and 19, according to the preferred embodiment;

Figure 17 is a flow diagram which follows "D" of Figure 12, according to the preferred embodiment;

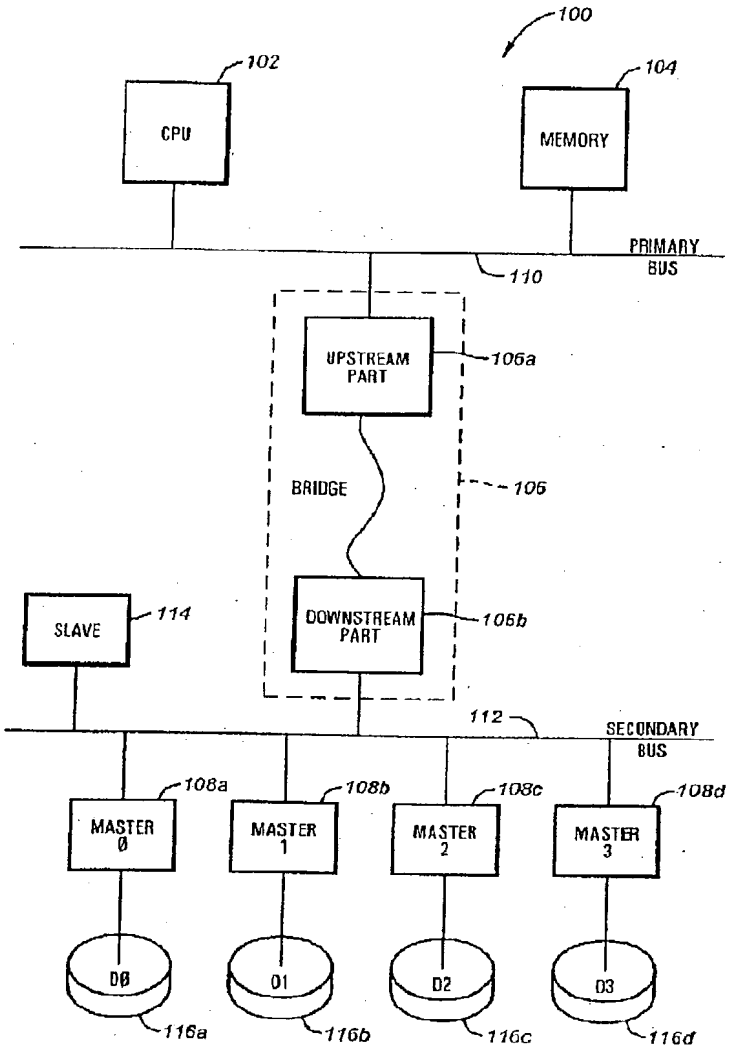
Figure 18 is a flow diagram which follows "E" of Figure 12, according to the preferred embodiment;

Figure 19 is a flow diagram which follows "F" of Figure 12, according to the preferred embodiment;

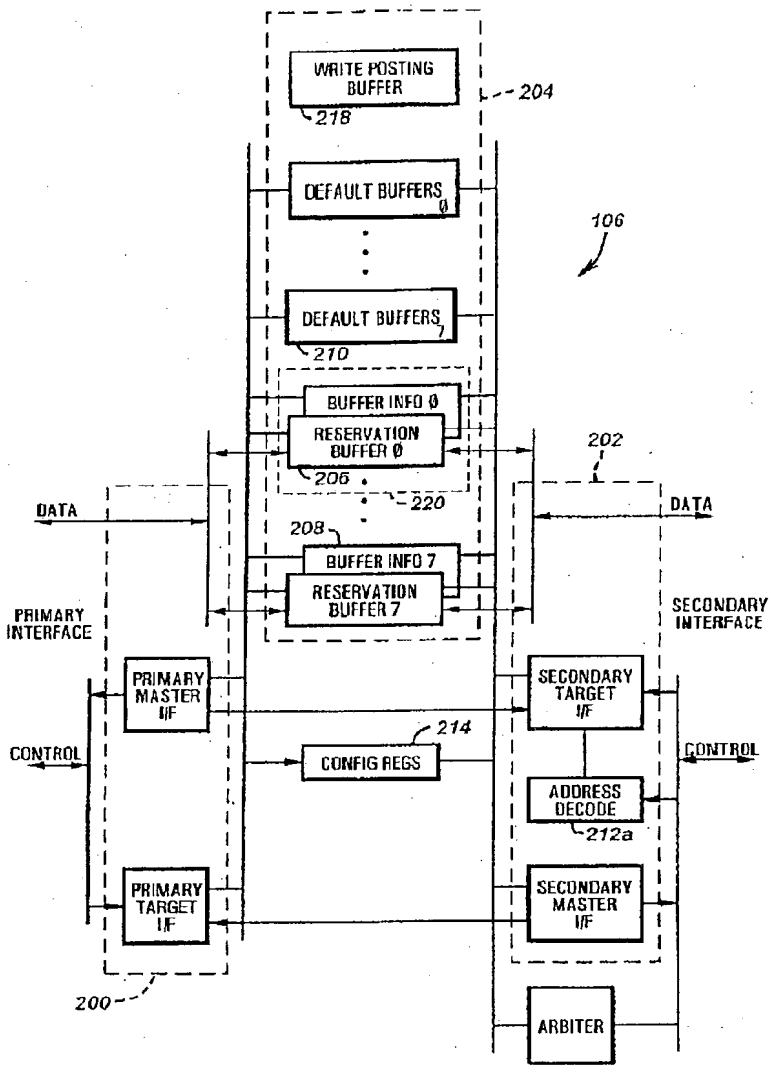
【図1】


100
102
CPU
104
MEMORY
PRIMARY
BUS
110
BRIDGE
106
112
SECONDARY
BUS
SLAVE
114
MASTER
108

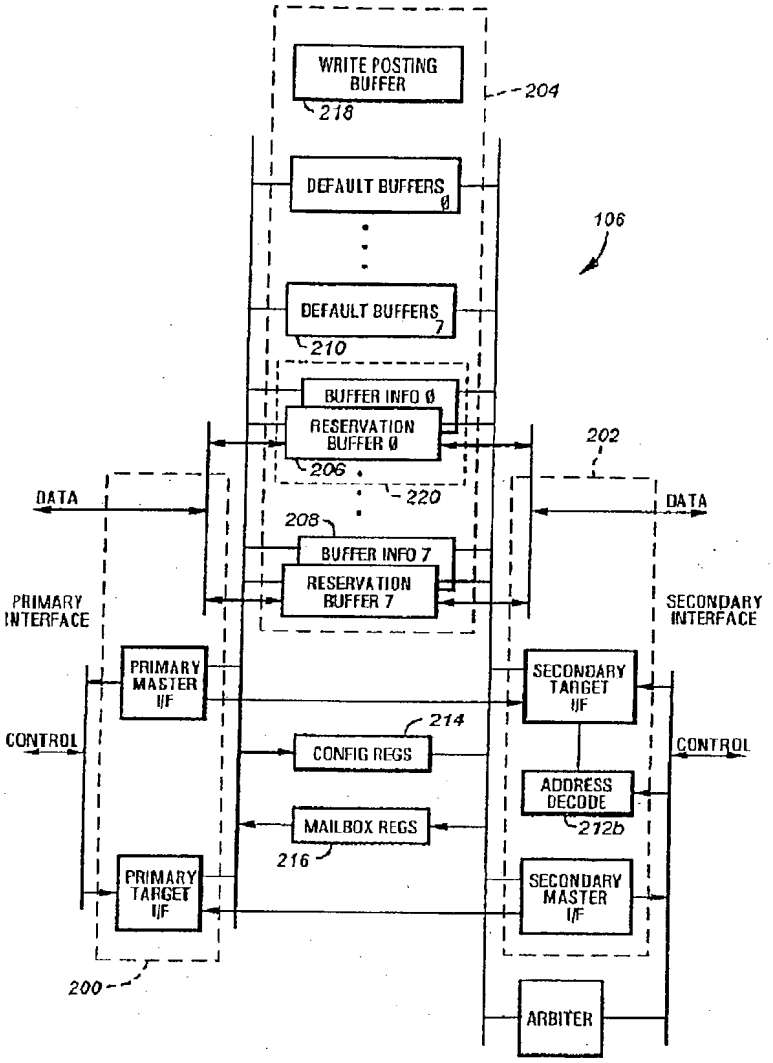
【図2】


100
102
CPU
104
MEMORY
PRIMARY BUS
110
UPSTREAM PART
106a
BRIDGE
106
DOWNSTREAM PART
106b
SLAVE
114
112
SECONDARY BUS
108a
MASTER 0
108b
MASTER 1
108c
MASTER 2
108d
MASTER 3
D0
D1
D2
D3
116a
116b
116c
116d

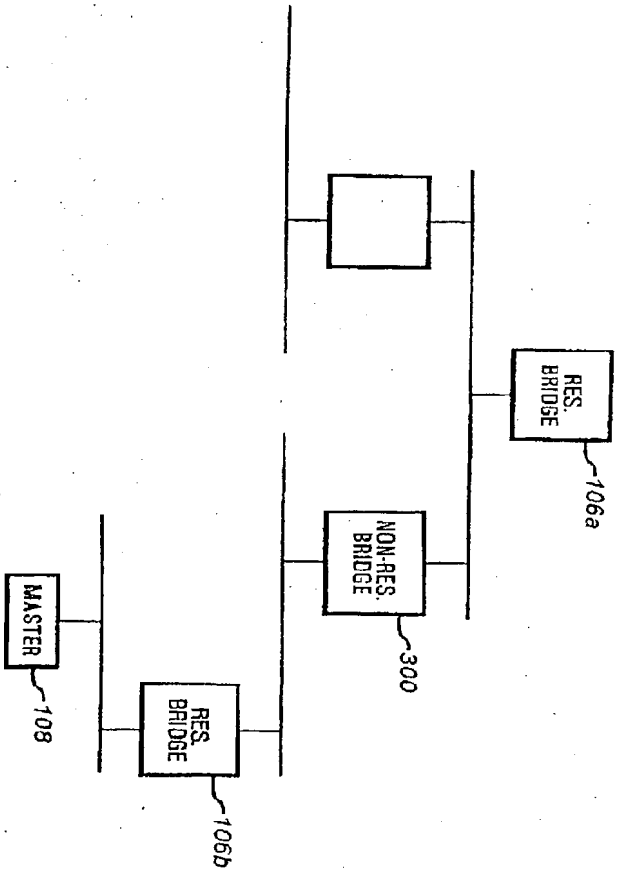
【図3】


WRITE POSTING BUFFER
218
204
DEFAULT BUFFERS 0
106
DEFAULT BUFFERS 7
210
BUFFER INFO 0
RESERVATION BUFFER 0
206
220
202
DATA
DATA
208
BUFFER INFO 7
RESERVATION BUFFER 7
PRIMARY INTERFACE
SECONDARY INTERFACE
PRIMARY MASTER I/F
SECONDARY TARGET I/F
214
CONTROL
CONFIG REGS
CONTROL
ADDRESS DECODE
212a
PRIMARY TARGET I/F
SECONDARY MASTER I/F
200
ARBITER

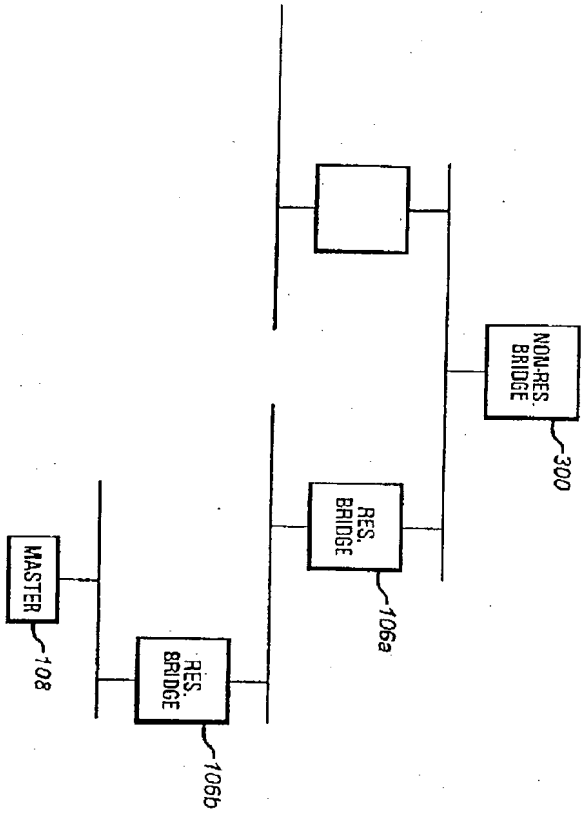
【図4】


WRITE POSTING BUFFER
218
204
DEFAULT BUFFERS 0
DEFAULT BUFFERS 7
210
106
BUFFER INFO 0
RESERVATION BUFFER 0
206
220
202
208
BUFFER INFO 7
RESERVATION BUFFER 7
DATA
DATA
PRIMARY INTERFACE
SECONDARY INTERFACE
PRIMARY MASTER I/F
SECONDARY TARGET I/F
214
CONFIG REGS
CONTROL
CONTROL
ADDRESS DECODE
212b
MAILBOX REGS
216
PRIMARY TARGET I/F
SECONDARY MASTER I/F
200
ARBITER

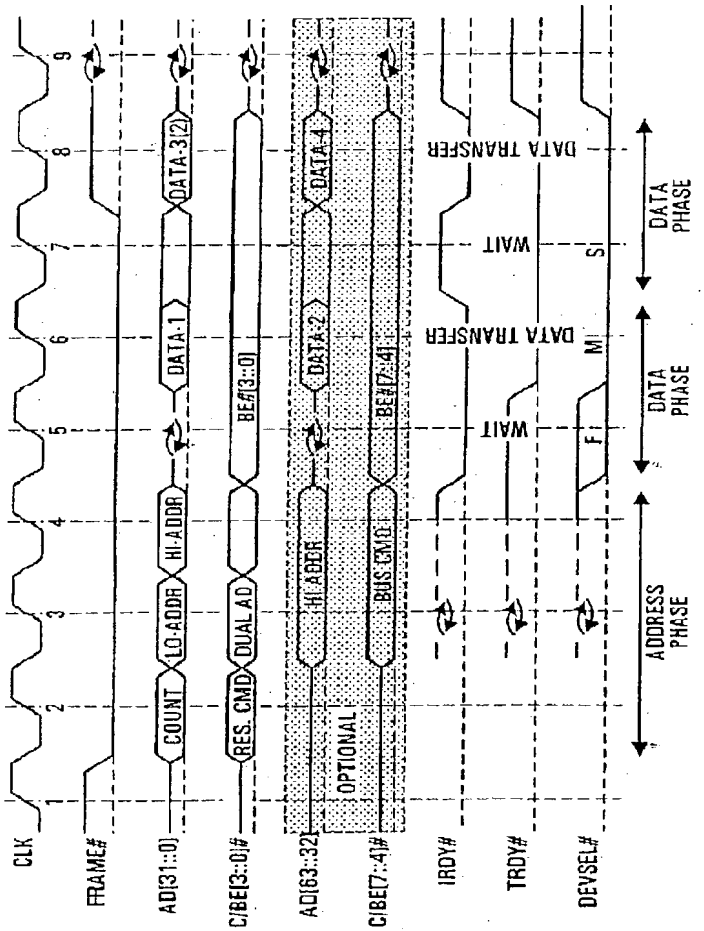
【図5】


RES. BRIDGE
106a
NON-RES. BRIDGE
300
MASTER
108
RES. BRIDGE
106b

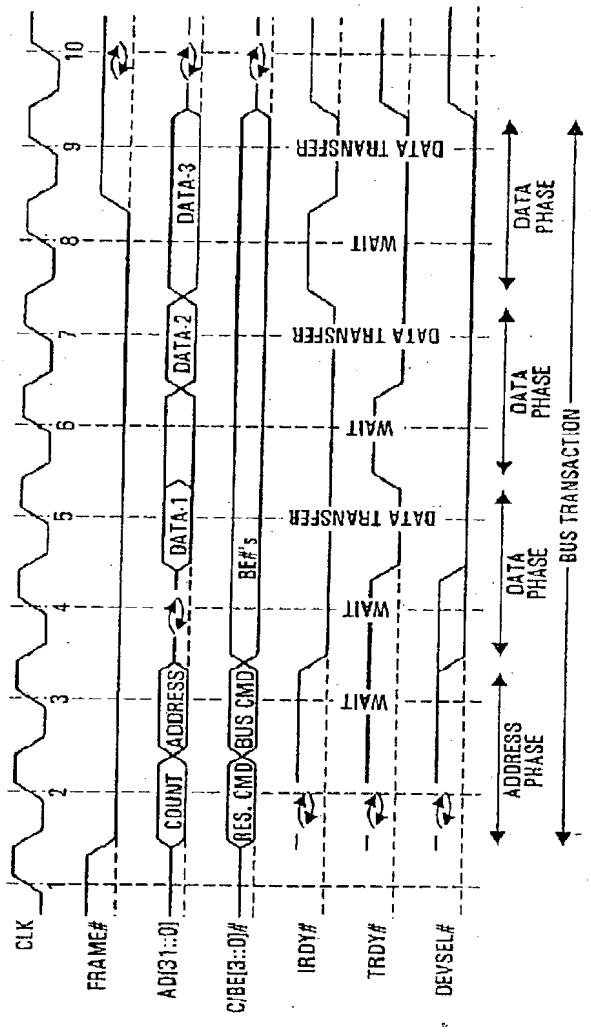
【図6】


NON-RES. BRIDGE
300
RES. BRIDGE
106a
RES. BRIDGE
106b
MASTER
108

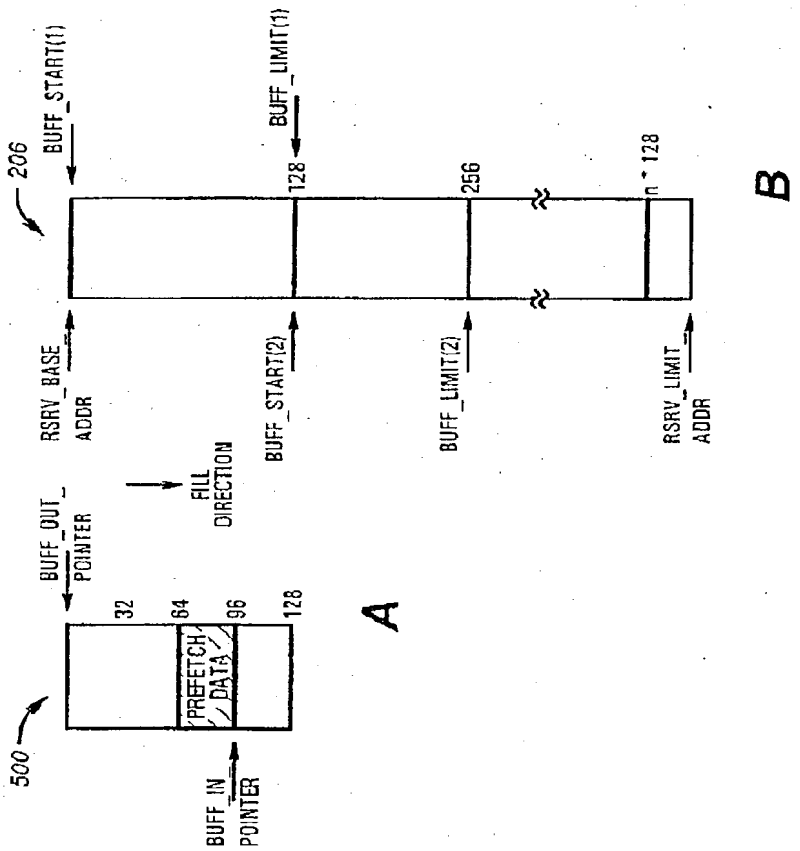
【図7】


CLK
FRAME#
AD[31::0]
C/BE[3::0]#
AD[63::32]
C/BE[7::4]#
IRDY#
TRDY#
DEVSEL#
1
2
3
4
5
6
7
8
9
COUNT
LO-ADDR
HI-ADDR
DATA-1
DATA-3(2)
RES. CMD
DUAL AD
BE#[3::0]
HI-ADDR
DATA-2
DATA-4
OPTIONAL
BUS CMD
BE#[7::4]
WAIT
DATA TRANSFER
WAIT
DATA TRANSFER
ADDRESS PHASE
DATA PHASE
DATA PHASE

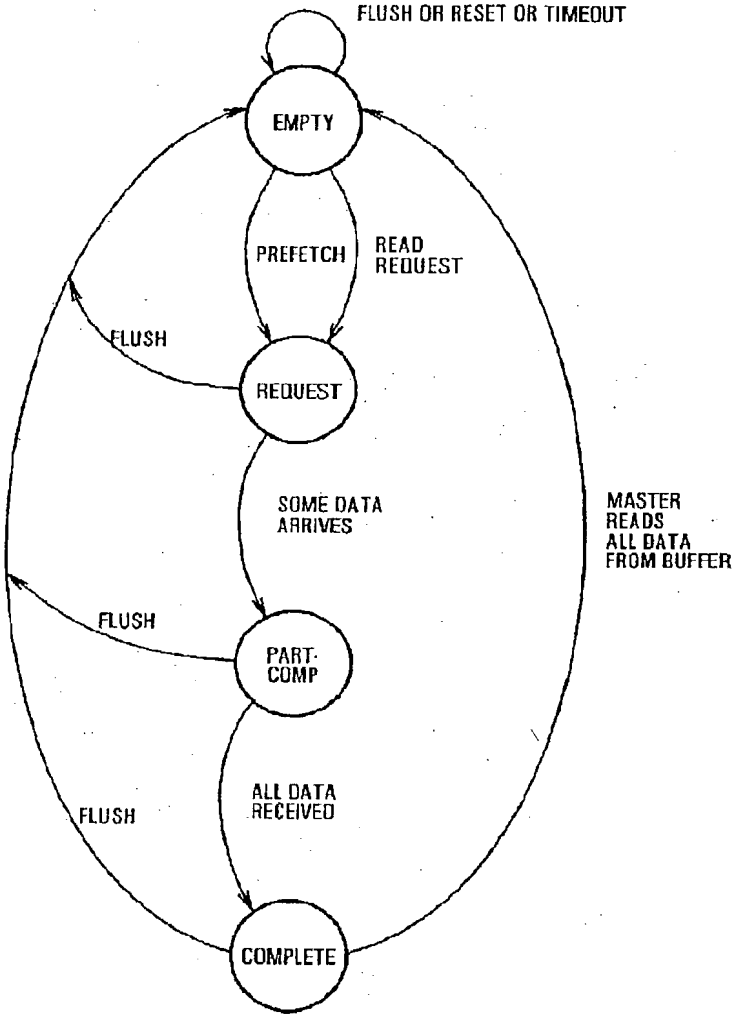
【図8】


CLK
FRAME#
AD[31::0]
C/BE[3::0]#
IRDY#
TRDY#
DEVSEL#
1
2
3
4
5
6
7
8
9
10
COUNT
RES. CMD
ADDRESS
BUS CMD
DATA-1
BE#'s
DATA-2
DATA-3
WAIT
DATA TRANSFER
ADDRESS PHASE
DATA PHASE
BUS TRANSACTION

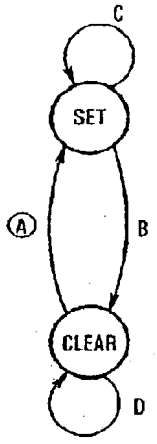
【図9】


206
BUFF_START(1)
BUFF_LIMIT(1)
128
256
n * 128
RSRV_BASE ADDR
BUFF_START(2)
BUFF_LIMIT(2)
RSRV_LIMIT ADDR
B
FILL DIRECTION
BUFF_OUT_ POINTER
32
64
96
128
PREFETCH DATA
A
500
BUFF_IN POINTER

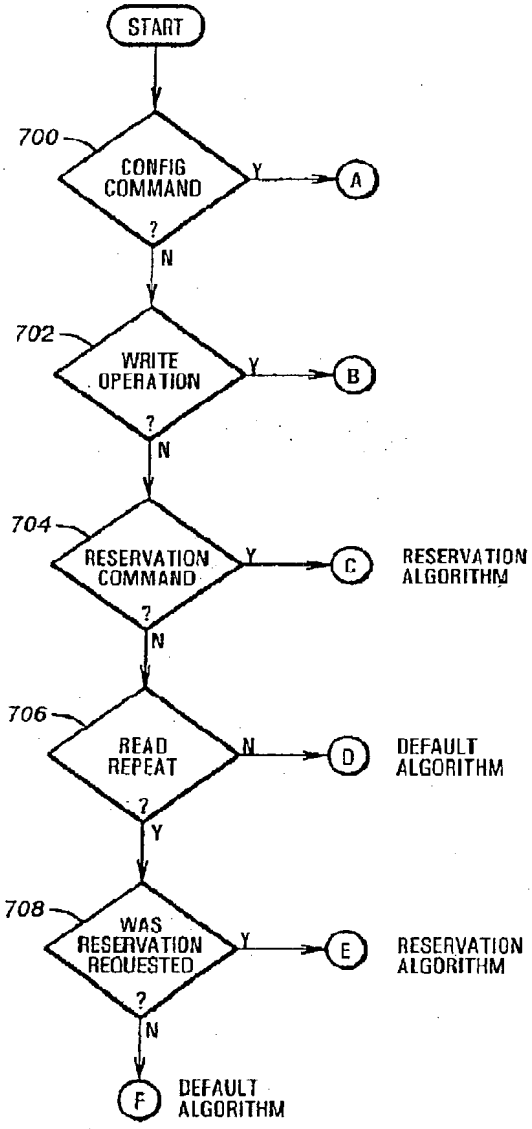
【図10】


FLUSH OR RESET OR TIMEOUT
EMPTY
PREFETCH
READ
REQUEST
FLUSH
REQUEST
SOME DATA
ARRIVES
MASTER
READS
ALL DATA
FROM BUFFER
FLUSH
PART-
COMP
ALL DATA
RECEIVED
FLUSH
COMPLETE

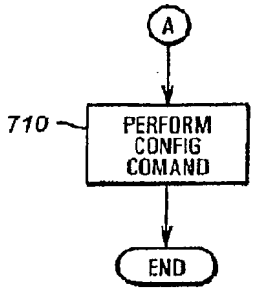
【図11】



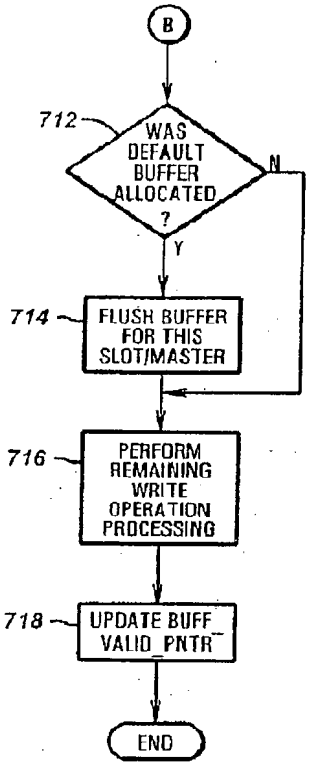
【図12】


START
700
CONFIG COMMAND
?
Y
A
N
702
WRITE OPERATION
?
Y
B
N
704
RESERVATION COMMAND
?
Y
C
RESERVATION ALGORITHM
N
706
READ REPEAT
?
N
D
DEFAULT ALGORITHM
Y
708
WAS RESERVATION REQUESTED
?
Y
E
RESERVATION ALGORITHM
N
F
DEFAULT ALGORITHM

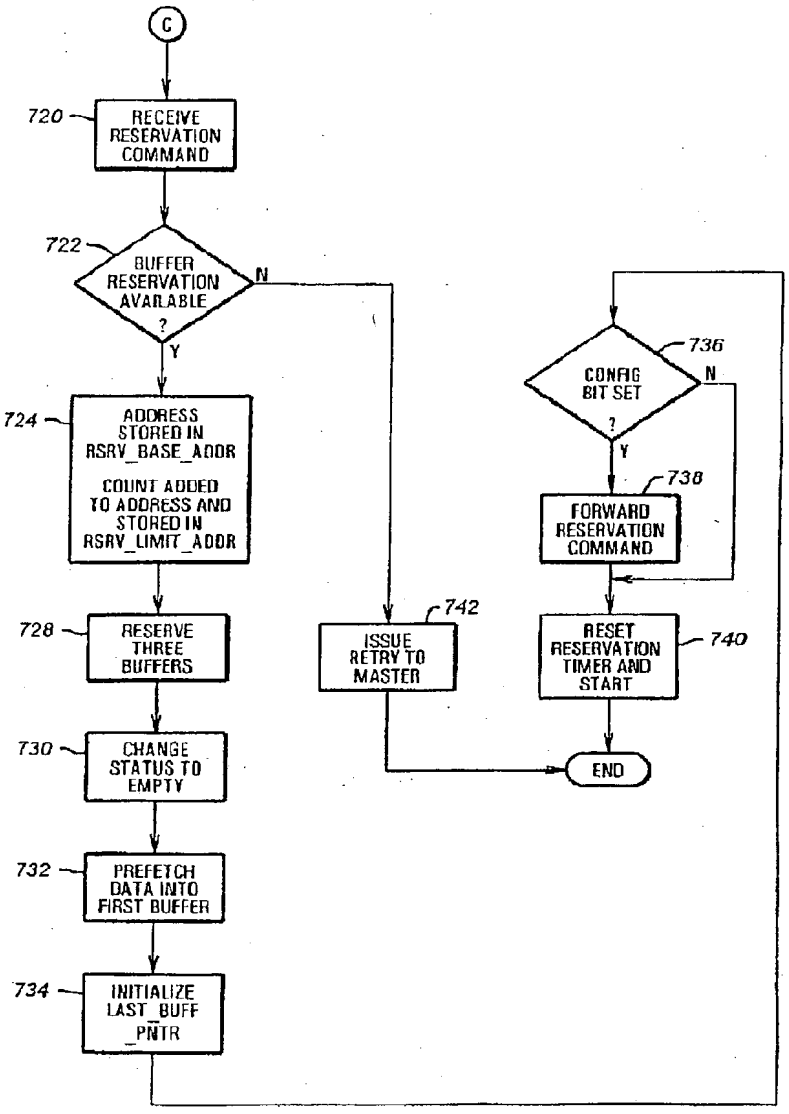
【図13】



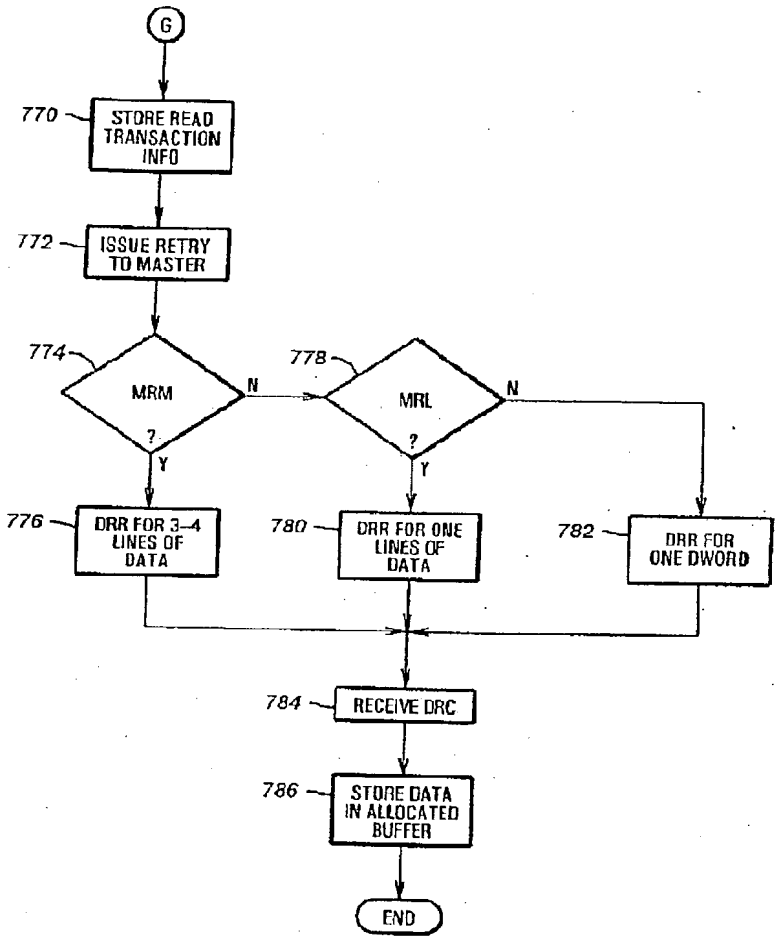
【図14】


B
712
WAS DEFAULT BUFFER ALLOCATED ?
N
Y
714
FLUSH BUFFER FOR THIS SLOT/MASTER
716
PERFORM REMAINING WRITE OPERATION PROCESSING
718
UPDATE BUFF_ VALID_PNTR
END

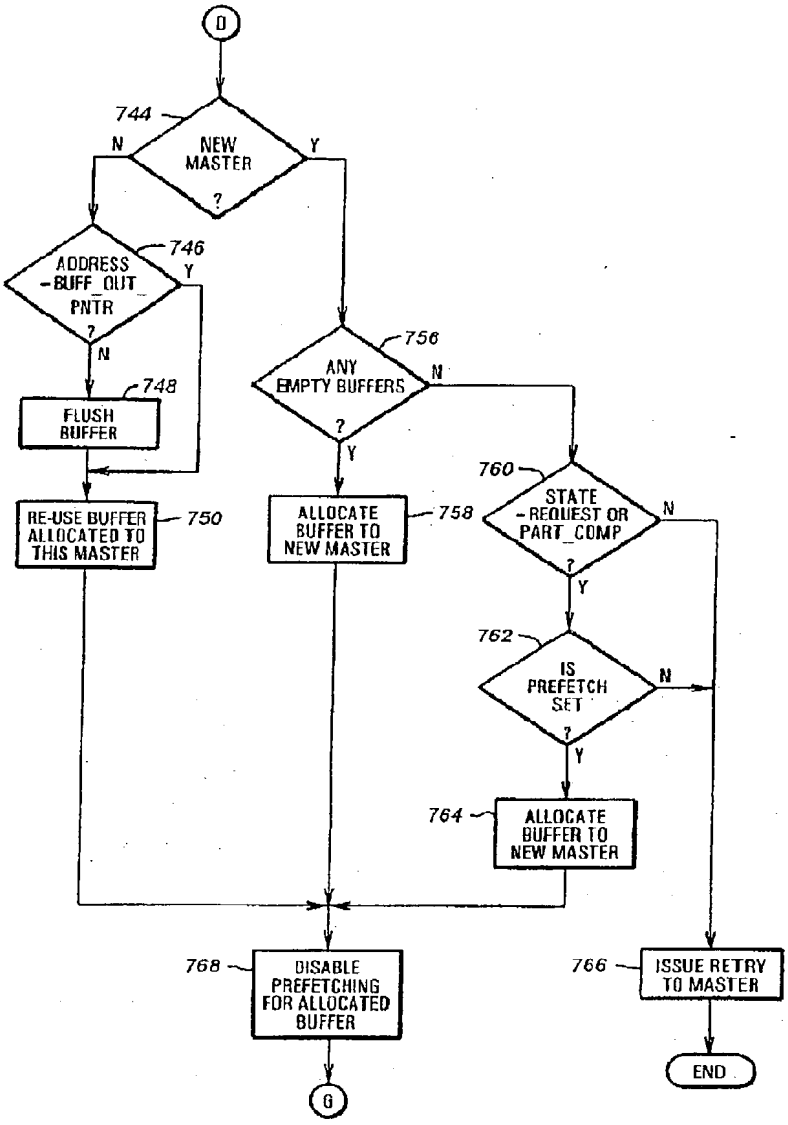
【図15】


C
720
RECEIVE RESERVATION COMMAND
722
BUFFER RESERVATION AVAILABLE ?
N
Y
724
ADDRESS STORED IN RSRV_BASE_ADDR
COUNT ADDED TO ADDRESS AND STORED IN RSRV_LIMIT_ADDR
728
RESERVE THREE BUFFERS
730
CHANGE STATUS TO EMPTY
732
PREFETCH DATA INTO FIRST BUFFER
734
INITIALIZE LAST_BUFF_PNTR
742
ISSUE RETRY TO MASTER
736
CONFIG BIT SET ?
N
Y
738
FORWARD RESERVATION COMMAND
740
RESET RESERVATION TIMER AND START
END

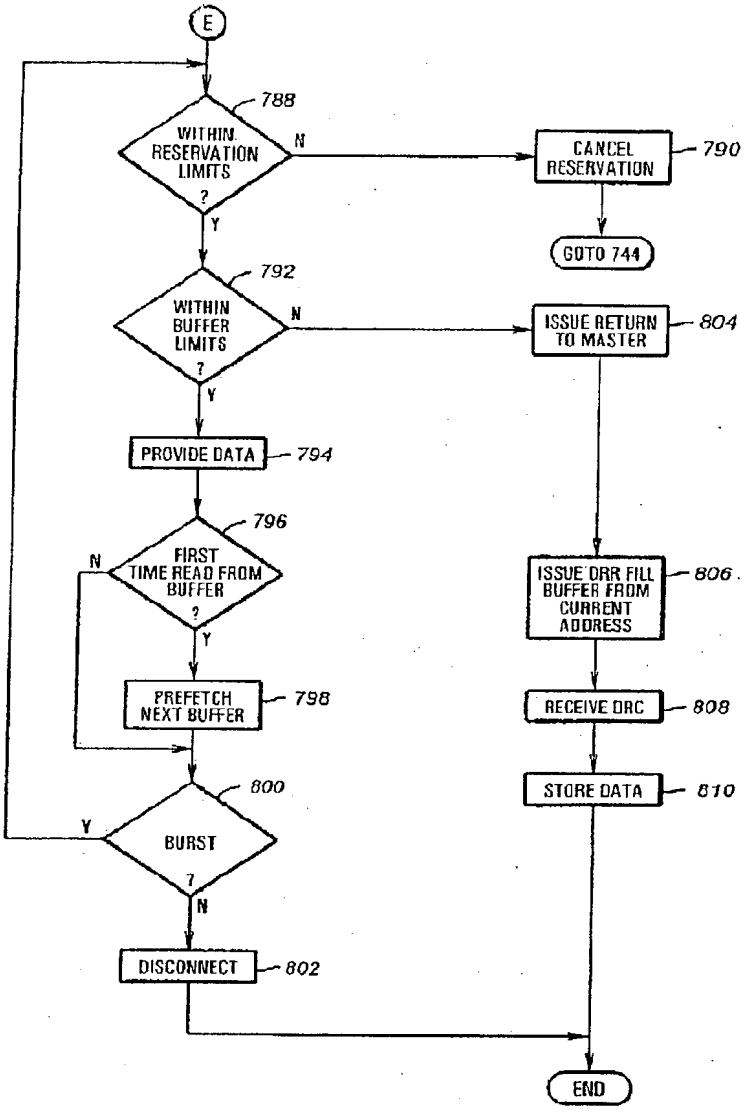
【図16】


G
770
STORE READ TRANSACTION INFO
772
ISSUE RETRY TO MASTER
774
MRM
?
N
Y
778
MRL
?
N
Y
776
DRR FOR 3–4 LINES OF DATA
780
DRR FOR ONE LINES OF DATA
782
DRR FOR ONE DWORD
784
RECEIVE DRC
786
STORE DATA IN ALLOCATED BUFFER
END

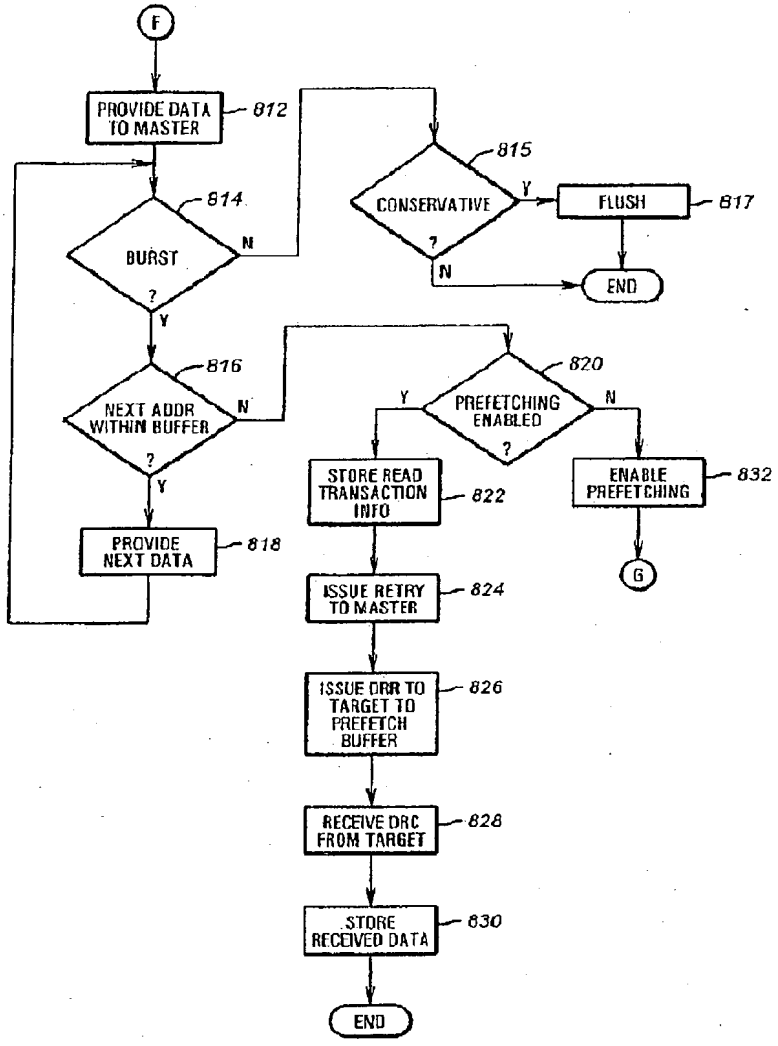
【図17】


D
744
NEW MASTER ?
N
Y
746
ADDRESS -BUFF_OUT_ PNTR ?
Y
N
748
FLUSH BUFFER
RE-USE BUFFER ALLOCATED TO THIS MASTER
750
756
ANY EMPTY BUFFERS ?
N
Y
ALLOCATE BUFFER TO NEW MASTER
758
760
STATE -REQUEST OR PART_COMP ?
N
Y
762
IS PREFETCH SET ?
N
Y
764
ALLOCATE BUFFER TO NEW MASTER
768
DISABLE PREFETCHING FOR ALLOCATED BUFFER
G
766
ISSUE RETRY TO MASTER
END

【図18】


E
788
WITHIN RESERVATION LIMITS ?
N
CANCEL RESERVATION
790
GOTO 744
Y
792
WITHIN BUFFER LIMITS ?
N
ISSUE RETURN TO MASTER
804
Y
PROVIDE DATA
794
796
FIRST TIME READ FROM BUFFER ?
N
ISSUE ORR FILL BUFFER FROM CURRENT ADDRESS
806
Y
PREFETCH NEXT BUFFER
798
RECEIVE DRC
808
STORE DATA
810
800
BURST ?
Y
N
DISCONNECT
802
END

【図19】


F
PROVIDE DATA TO MASTER
812
814
BURST ?
N
Y
815
CONSERVATIVE ?
Y
FLUSH
817
N
END
816
NEXT ADDR WITHIN BUFFER ?
N
Y
PROVIDE NEXT DATA
818
820
PREFETCHING ENABLED ?
Y
N
STORE READ TRANSACTION INFO
822
ENABLE PREFETCHING
832
G
ISSUE RETRY TO MASTER
824
ISSUE DRR TO TARGET TO PREFETCH BUFFER
826
RECEIVE DRC FROM TARGET
828
STORE RECEIVED DATA
830
END

1. Abstract

A method for transferring data through a bus bridge. The bus bridge includes a number of data buffers for storing data, prefetching data and write posting data. A device communicating with the bus bridge may reserve a buffer by one of two reservation mechanism. The reservation mechanism provides the bus bridge with the address and byte count. The reservation may also be forwarded to any upstream bus bridges. The reserved buffers are prefetched for efficient use of bus access. Data is prefetched and flushed according to alternative algorithms if a buffer is not reserved.

2. Representative Drawing
Figure 2